Algorithmic Techniques in Algebraic Cryptanalysis and their Applications

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr. S. de Rijcke,
volgens besluit van het college voor promoties
te verdedigen op woensdag 27 mei 2026
klokke 14:30 uur

door

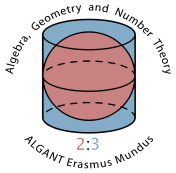
Rusydi Hasan Makarim
geboren te Cilacap, Indonesië
in 1989

Promotor:
Prof.dr. R. Cramer (CWI Amsterdam & Universiteit Leiden)

Co-promotor:
Dr.ir. M. Stevens (CWI Amsterdam)

Promotiecommissie:
Prof.dr.ir. G.L.A. Derks
Prof.dr. P. Stevenhagen
Prof.dr. C. Cid (Simula UiB, Norway & OIST, Japan)
Prof.dr. A. May (Ruhr-University Bochum, Germany)
Dr. S. Samardjiska (Radboud Universiteit, Nijmegen)

The research was carried out in the Cryptology Group at Centrum Wiskunde & Informatica (CWI) Amsterdam. The PhD position was largely funded by the ALGANT-DOC Erasmus-Mundus program and partially funded by the ERC Advanced Grant 740972 (ALGSTRONGCRYPTO).



Algorithmic Techniques in Algebraic Cryptanalysis and their Applications

Rusydi H. Makarim

Contents

Part I
Introduction and Overview of the Results

1 Cryptography and Cryptanalysis

Contents

1.1 Cryptography

Communication is one of the most basic human activities, enabling two or more parties to exchange messages through various forms of media. Traditionally, communication could only be achieved within a very limited distance by means of meeting or gathering in one place. When the concept of writing was invented, especially with portable mediums such as animal skins, bones, papyrus, etc, information could reach longer distances by writing the message on a suitable material and send it to the destination using a courier. If the message contains sensitive information, the courier is assumed to be a trusted person that would not leak it to malicious parties. In order to liberate the sender and receiver from such assumptions, they need to devise techniques that guarantee the secrecy of their message.

Classically, cryptography is the practice and study of techniques for secure information processing in the presence of adversaries. This definition is relevant when it comes to the study of encryption schemes and digital signatures, both are used to ensure secrecy and authenticity of a communication channel. Modern cryptography, on the other hand, has a broader context. It is a study how to obtain cryptographic functionalities such as confidentiality, integrity, entity authentication, and data origin authentication [MvOV96], typically using mathematical techniques related to various aspects of information security. For instance, confidentiality is achieved using encryption primitives that protect the data by converting it into an incomprehensible form that can be recovered only by an authorized entity; integrity is ensured through mechanisms that detect any unauthorized modification of information; entity authentication verifies the identity of a communicating party and is commonly achieved using challenge-response protocols, digital certificates, or public-key infrastructures; data origin authentication ensures that a received message originates from the claimed source using message authentication codes or digital signatures. A further example includes the study of secure computation, which allows arbitrary computation to be distributed among multiple parties such that the computations remain confidential and performed correctly even if some parties involved behave maliciously.

Encryption is a function that transforms a message (plaintext) into its corresponding incomprehensible form of information (ciphertext). The correspondence between a plaintext and a ciphertext is necessary in order for the intended

recipient to recover the original plaintext using a *decryption* function. To provide security against adversaries, the encryption and decryption function of an encryption scheme require an additional secret information as input, which should only be available to the designated parties. We refer to this secret input as *key*. A classical construction of an encryption scheme is done storing every possible plaintext and their corresponding ciphertext in a look-up table. Another well-known example is Caesar cipher (named after Julius Caesar), that takes a secret number N as a key and the encryption function substitutes each letter in a message by the next N-th letter in a cyclic and alphabetical ordering (i.e., with $N = 3$, A is substituted by D, B by E, ..., W by Z, X by A, Y by B, and Z by C).

The encryption and decryption functions of an encryption scheme should be invertible and efficiently computable, provided that the key is known. In principle the security of an encryption scheme must only rely on the secrecy of the key, not the secrecy of the scheme itself. This is known as the *Kerckhoffs' principle* [Ker83]. The size of a key should be small enough to ensure low storage distribution cost. However, given a plaintext and its corresponding ciphertext, the number of all possible keys should be large enough such that no real-world adversary is able to exhaustively try all possible keys. The usage of keys is also beneficial whenever an adversary succeeds in revealing some information about the key. Instead of redesigning the entire scheme, the legitimate parties can simply replace the previous keys with the new ones.

Encryption schemes can be classified according to the relation between the encryption key and decryption key. There are two main categories. The first category is *symmetric-key* (or secret-key) cryptography. The name comes from the fact that the same key is used to perform encryption and decryption. Thus, the key must not be accessible by adversaries. The second category is *asymmetric-key* (or public-key) cryptography. The idea is that, given an encryption key, it should be computationally infeasible for an adversary to obtain the decryption key. This allows the knowledge of the encryption key to be known publicly. The idea of public-key cryptography was formally introduced in [Mer78, DH76].

There are a number of approaches to construct symmetric-key encryption schemes. One particular class of symmetric-key cryptography are *block ciphers*. As the name suggests, a block cipher operates on a fixed-length block of plaintext. They are widely used on applications that require encryption on a bulk of data. Modern block ciphers typically employ an iterated function called *round function*. It is constructed as a composition of a nonlinear function together with a linear function and key addition operation. The goal of employing a nonlinear function is to make the ciphertext statistics depend on the plaintext statistics in a way that makes it difficult to be exploited by an adversary. This design principle is known as *confusion*. A linear function and key addition are used so that each bit of the plaintext and each bit of the key influences many bits of the ciphertext. This design principle is known as *diffusion*. Both principles were first introduced by Shannon [Sha49]. Some notable examples of modern block ciphers are Data Encryption Standard (DES) [Nat77] and Advanced Encryption Standard (AES) [NIS01]. In order to encrypt an arbitrary length plaintext using a block cipher, one uses modes of operation, i.e., it is an algorithm that specifies how a block cipher should be applied to achieve this. Some examples of block cipher modes of operation include Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Cipher Feedback (CFB) (see [Pre11]).

Another class of symmetric-key cryptography are *stream ciphers*, where encryption and decryption functions operate on one bit at a time. They are preferred over block ciphers in devices with limited memory and each plaintext bit must be processed individually. An example of a simple, yet secure, stream cipher is the *one time pad*. Given a sequence of bits $s_1, s_2, \ldots \in \mathbb{F}_2$ called *keystream*, the one-time pad encrypts a sequence of plaintext bits $p_1, p_2, \ldots \in \mathbb{F}_2$ into a sequence of ciphertext bits $c_1, c_2, \ldots \in \mathbb{F}_2$ as $c_i = p_i + s_i$ for $i \geq 1$. It achieves perfect secrecy in the sense that a ciphertext does not reveal any information about the plaintext provided the same keystream is not used twice to encrypt different plaintext. One drawback of the one-time pad is that the length of the keystream must be equal to the length of the plaintext. A solution for this drawback at the cost of losing perfect secrecy is to let a pseudo-random generator generate the keystream. It takes a relatively short key as its initial state together with some public parameters. The study of stream ciphers deals with techniques to construct secure pseudo-random generator that generates the keystream with low cost digital circuit operations.

One of the main building blocks of stream ciphers is a *Linear Feedback Shift Register* (LFSR). An LFSR of length ℓ over $\mathbb{F}_2$ is a regularly clocked finite state machine with internal state $(x_1, \ldots, x_\ell) \in \mathbb{F}_2^\ell$ and an update function $L(x_1, \ldots, x_\ell) = (x_2, \ldots, x_\ell, \Lambda(x_1, \ldots, x_\ell))$ where $\Lambda : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is a linear map. At each clock, it outputs a specific entry of its internal state, followed by the update function. Let $S_t \in \mathbb{F}_2^\ell$ denote the internal state at time t. The internal state at time $t+1$ is computed as $S_{t+1} = L(S_t) = L^{t+1}(S_0)$ where S_0 denote the initial state of the LFSR. Note that L itself is a linear transformation. Thus, an LFSR can not be used directly as a keystream generator because the knowledge of the entire keystream can be efficiently recovered, provided that Λ is known and an attacker has access to χ consecutive keystream bits, where χ is the size of the smallest LFSR that generates the given sequence (also known as *linear complexity* of the sequence). A common technique to construct a stream cipher is to combine one or several LFSRs together with a nonlinear Boolean function in order to increase the complexity of attacks that exploit algebraic relations between the keystream bits and the initial state of the stream cipher.

The main problem of using symmetric-key encryption schemes for communication purposes is that the keys must be distributed to the intended recipients via a secure channel. This limitation was solved with the introduction of public-key cryptography. Recall that the encryption keys of a public-key encryption scheme can be publicly known and only the decryption keys that must be kept secret. This idea is possible due to the concept of a *trapdoor one-way function*. Informally, it is a function that is computationally feasible to compute in one direction yet it is computationally infeasible to compute its inverse unless one knows some secret information called the *trapdoor*. One prominent example of trapdoor one-way functions is RSA [RSA78], that is based on the difficulty of integer factorization. The trapdoor consists of prime factors of a sufficiently large positive composite integer. Other examples of trapdoor one-way functions include the knapsack problem [MH78], discrete logarithm problem [DH76, Gam84, Kra93], lattice problems [BDK+18, DKL+18, HPS98, PFH+] and multivariate quadratic (MQ) problem [Pat96, KPG99, PCDY17, CBD+09].

Such a trapdoor one-way function also enables a party to attest his approval of a message, analogous to the concept of signing a document in a real-world scenario. This mechanism is known as a *digital signature scheme*. A digital

signature scheme consists of three different algorithms: *key generation*, *signing*, and *verification*. The key generation algorithm is responsible for generating a private-key and its corresponding public-key. The signing algorithm produces a signature for a particular message using a private-key whereas the verification algorithm verifies the signature of a given message using a public-key.

1.2 Cryptanalysis

Cryptanalysis is the process of developing methods to attack cryptographic schemes by unveiling and exploiting their intrinsic properties. In the case of encryption schemes, the aim is to devise an algorithm that breaks one of the claimed security properties. The algorithm can be generic and applicable to all encryption schemes that belongs to a specific family, or it can be dedicated for a particular construction. Typically, an attack scenario assumes that the encryption or decryption algorithm is known to the adversaries, and the key used is chosen uniformly at random from the set of all possible keys.

There are different goals for cryptanalytic attacks. The strongest one is to extract partial or full information about the key used in an encryption. This would allow an adversary to recover any plaintext encrypted using the same key. A weaker goal of cryptanalytic attack is to learn non-random behaviours of a particular encryption scheme, known as *distinguishing attacks*. For block ciphers, distinguishing attacks try to exhibit some properties that deviates from a random permutation, while for the case of stream ciphers, it tries is to distinguish the keystream from a truly random sequence. In a digital signature scheme, an attack that tries to falsify a signature for a given message without an access the the actual signer's private key is called a *forgery attack*.

In any cryptanalytic attack, it is possible to assume that adversaries have access to various types of data (e.g., plaintext, ciphertext, etc). Not all data is equally beneficial. Thus it is crucial to classify different attack scenarios based on the data required to mount the attack. Let E_K and D_K denote encryption and decryption function under the key K.

Ciphertext-Only Attack In this model an adversary is assumed to only have access to a set of ciphertexts.

Known-Plaintext Attack In this attack model an adversary has access to a set of, say, n plaintext $p_1, p_2, \ldots, p_n$ and their corresponding ciphertext $c_1, c_2, \ldots, c_n$.

Chosen-(Plaintext/Ciphertext) Attack In a chosen-plaintext attack an adversary selects a set of plaintext $p_1, \ldots, p_n$ *a priori* and requests their corresponding ciphertext $c_i = E_K(p_i)$ for all $i = 1, 2, \ldots, n$. Conversely, in a chosen-ciphertext attack an adversary chooses a set of ciphertext $c_1, \ldots, c_n$ and requests their corresponding plaintext $p_i = D_K(c_i)$ for all $i = 1, 2, \ldots, n$.

Adaptively Chosen-(Plaintext/Ciphertext) Attack In an adaptively chosen-plaintext attack, an adversary chooses plaintexts for encryption based on previously selected plaintexts and the corresponding ciphertexts. In an adaptively chosen-ciphertext attack the plaintext and encryption oracle in the adaptively chosen-plaintext scenario are replaced by a ciphertext and decryption oracle, respectively.

Ciphertext-only and known-plaintext attacks are classified as *passive* attacks, i.e., the adversary only observes communications or data. The former is considered to be the most realistic attack scenario but it is also the most difficult one due to limited information available to the adversary. Thus, to recover partial or full-information about the key in a ciphertext-only attack, additional knowledge about the plaintext is often needed, such as some redundancy that occurs in the plaintext or knowledge of a valid plaintext.

On the other hand, chosen-(plaintext/ciphertext) and adaptively chosen-(plaintext/ciphertext) attacks are *active* attacks. These models assume that the adversary has black-box access to the encryption (or decryption) oracle with the user's key, and therefore viewed as somewhat less realistic in practice. In public-key cryptography, however, the encryption oracle comes for free since the encryption key is public.

As we move down the list above, the requirements of the attack model become increasingly difficult. This can be seen from the escalation in the type of data required and the capability of the adversary. In order to compare different types of attack, it is necessary to quantify their complexity in terms of all involved parameters. The cost metric for a cryptanalytic attack consists of the following parameters:

Time complexity In general, the time required to mount the attack is considered the most important cost metric in cryptanalysis. In symmetric-key cryptanalysis, it is typically quantitatively measured relative to the cost of an encryption or decryption call. In public-key cryptanalysis, the time complexity is typically measured in terms of the number of bit operations.

Data complexity Every attack model previously described requires certain information to be available. Not only the type of information, the amount must also be taken into account as a cost metric for cryptanalytic attacks. There is also a possible trade-off between data complexity and time complexity. For instance an attack might need relatively few encryption or decryption operations but it requires enormous amounts of plaintext or ciphertext pairs, or *vice versa*.

Memory complexity The memory complexity is a metric that represents the storage requirements to mount the attack. High memory complexities may turn out to be the main bottleneck that causes an attack to be impractical. It is also considered to be an expensive cost from practical point of view. In the case where there is a trade-off between time complexity and memory complexity, it is often more reasonable to reduce the memory complexity at the expense of increasing time complexity.

Success probability In all attack settings, it is plausible to include success probability as a cost metric to mount an attack. For instance, the adversary may reduce the data complexity at the expense of a smaller success probability. However, the probability should still be sufficiently high to be considered as a valid attack.

The most straightforward cryptanalysis technique that recovers the key is *exhaustive key search*. The goal is to recover the key used to encrypt a particular ciphertext, which will allow an adversary to recover the corresponding plaintext and any other ciphertexts encrypted with the same key. It requires only one

ciphertext and sufficient knowledge about the plaintext to eliminate wrong keys during the searching process. Exhaustive key search works for all encryption schemes independent from their specification. Thus, it constitutes as a baseline to determine whether a specific algorithm can be considered as a structural attack. More formally, an algorithm is classified as a break of an encryption scheme if its time complexity is strictly less than exhaustive-key search. Note that it does not necessarily mean that a scheme that is broken in such sense is considered to be broken in practice.

One of an early known documented cryptanalysis technique in history is attributed to *Al-Kindi* (Alkindus) [Sin00, MAAT02][vzG15, Section G.1]. His treatise, dated back in the 9-th century, described a method to recover the plaintext from a ciphertext encrypted using *monoalphabetic substitution cipher*, i.e., it is an encryption scheme that substitutes each plaintext character into another character/symbols and the rule of the substitution is unknown to adversaries. The only requirement of the attack is that the original language used in the communication is known. The main idea of the attack is the occurrences of a letter or combination of letters for a particular language varies. For instance, the letters 'E', 'T', and 'A' are the three most-frequent letters that appear in English language while occurrence of 'X', 'Q', or 'Z' is rare. An adversary then counts the frequency of each symbol in the ciphertext and associates each of them with a corresponding letter of the language used such that the occurrence falls within the same range. This can also be combined with more sophisticated technique such as counting a pair of letters (*bigram*) or triplet of letters (*trigram*). This technique, which is statistical in nature, is known as *frequency distribution analysis*.

Many of the cryptanalysis approaches in modern cryptography, especially in the domain of symmetric-key cryptanalysis, are also based on statistical techniques. For instance, *linear cryptanalysis* [Mat93] and *differential cryptanalysis* [BS90] are the two most-well known cryptanalysis techniques for block ciphers exploiting non-random statistical behaviour of the encryption function. In linear cryptanalysis, an adversary tries to identify linear relations among parity bits of the plaintext, the ciphertext, and the key* that holds with high or low probability. On the other hand, differential cryptanalysis studies how the relation between two plaintexts affects the probabilistic occurrence of a certain relation in the corresponding ciphertexts. Both attacks are also applicable to stream ciphers [Gol94, BD07]. Another type of statistical attack on stream ciphers consists of *correlation attacks* [Sie85]. It is applicable to any stream cipher where the keystream is generated by combining the output of several LFSRs using a nonlinear function. A correlation attack exploits possible statistical dependence between the keystream bits and the output of a single LFSR. A more efficient variant of it is described in [MS89a].

One of the main drawbacks of statistical cryptanalysis is that it often requires large amounts of plaintexts or ciphertexts, making it less realistic in practice to mount the attack. Moreover, some statistical assumptions are required in order to simplify the analysis. For instance, the assumption of independent round keys is often stated in an attack against block ciphers based on linear or differential cryptanalysis. This assumption does not hold true in general, but

* Concretely, the relation is of the form $\sum_{i=1}^{n}(\alpha_i P_i + \beta_i C_i) = \sum_{j=1}^{k} \gamma_j K_j$ where $\alpha_i, \beta_i, \gamma_j \in \mathbb{F}_2$ and P_i, C_i, K_j are the plaintext bits, the ciphertext bits, and the key bits respectively

often is an useful heuristic.

Another approach in cryptanalysis is *algebraic cryptanalysis* or *algebraic attack* [Bar09]. From a high-level point-of-view, algebraic cryptanalysis consists mainly of two steps. It first converts the encryption or decryption function of an encryption scheme into a system of multivariate equations with coefficients in some ring or field such as $\mathbb{F}_2$. The indeterminates of the system of equations represent the internal state of the encryption function and the key, both unknown to the adversaries. The second step of the attack is to solve the constructed system of equations for a given plaintext and its corresponding ciphertext. Solving the system of equations means recovering the key as well as the internal state of the cipher. Note that for a high-degree system of equations, a common strategy to reduce the degree is by substituting all quadratic terms with auxiliary variables. For instance, a term $x_i x_j$ is substituted by $y_{i,j}$ and the new equation $x_i x_j - y_{i,j} = 0$ is added to the system of equations [KS99, CKPS00]. By iteratively applying such strategy, one can always reduce the degree of a system of equations to a quadratic one at the expense of having more variables and equations compared to the initial system of equations.

Algebraic cryptanalysis serves as a complementary technique to statistical cryptanalysis. Its primary advantage is that it requires only few plaintext-ciphertext pairs to mount the attack. In contrast to the probabilistic nature of statistical cryptanalysis, algebraic cryptanalysis is a deterministic technique and the success probability of the attack is always guaranteed since it requires no *a priori* assumptions.

However, one drawback of algebraic cryptanalysis is that it is computationally hard to solve a system of quadratic polynomial equations over a finite field, even in the case of binary field $\mathbb{F}_2$. Solving a quadratic system of equations is a well-known hard problem, known as the *Multivariate Quadratic (MQ) problem*. The computational intractability of the MQ problem over a finite field is also used to construct trapdoor functions for public-key cryptography. We refer to the family of such construction as *multivariate public-key cryptography* (MPKC), which also includes digital signature schemes based on the hardness of MQ problem. The primary challenge in MPKC is to construct a public-key or digital signature scheme such that the complexity to solve the system of equations representing the public-key is as close as possible to the complexity of solving random quadratic polynomial equations.

MPKC is one of the mathematical platforms considered for post-quantum cryptography. The US National Institute for Standards and Technology (NIST) has initiated a competition for a post-quantum cryptography standardization process. The goal is to develop classical cryptographic schemes that are compatible with existing network/communication protocols and considered secure against attacks by both quantum and classical computers. The security of several submitted proposals are based on the hardness of the MQ problem over finite fields.

1.3 Contributions

1.3.1 Motivations

This thesis focuses on the development of Gröbner bases algorithms and their applications in the cryptanalysis of Multivariate Public-Key Cryptography (MPKC)

and in other domains. The main research question in this work is how efficient the Gröbner bases algorithms can be implemented in practice to solve a system of equations that represent the public-key of MPKC. In this section we shall discuss the motivations on why we focus on Gröbner bases algorithms and why we concentrate on their applications in the cryptanalysis of MPKC. While Gröbner bases algorithms and polynomial-solving methods are often associated with the cryptanalysis of MPKC, they are equally applicable in the analysis of symmetric-key primitives. These primitives can be expressed as systems of polynomial equations over finite fields, which allows the use of Gröbner bases algorithms to perform key-recovery attacks or to study the underlying algebraic structures under certain conditions. Chapter 2 provides an overview of these approaches.

In the view of algebraic cryptanalysis, there are two factors that influence its effectiveness to attack cryptographic primitives. The first factor is the representation of the primitive as a system of multivariate polynomial equations. An adversary needs to find an algebraic modelling that maximizes the efficiency of algorithms that solve it. For instance, when it comes to algebraic solvers such as Gröbner bases algorithms [Buc65, Fau99] or the XL algorithm [CKPS00], it is more desirable that the system contains as many linearly independent equations as possible (see Subsection 2.5.1 and Subsection 2.5.2). The efficiency of other methods may be influenced by other parameters, such as the Raddum-Semaev algorithm that takes the sparsity of the system and the order of the finite field into account (see Subsection 2.5.4). The second factor, which is arguably the most important one, is the algorithm used to solve the system of polynomial equations. The SAT-based approach is applicable for polynomials with coefficients in $\mathbb{F}_{2^e}$ where e is a positive integer (see Subsection 2.5.3 and Chapter 15 of [Bar09]). The Raddum-Semaev algorithm is applicable for sparse systems of polynomial equations defined over a finite field having small order. The MILP-based approach is so far limited only for polynomials defined over $\mathbb{F}_2$ (see Subsection 2.5.5). Gröbner bases algorithms and linearization-based algorithms remain as the most generic approaches. The latter approach arguably received the most attention in cryptographic literature. Since the introduction of the XL algorithm [CKPS00], several modified variants were proposed [CP02, CP03, Cou04]. Even after it was shown that the XL algorithm is a redundant variant of the F_4 algorithm [AFI+04], the development of XL-based algorithms still continued with the introduction of the concept of mutant and MutantXL algorithms [MMDB08, MCD+09, MDBW09]. Eventually, the MutantXL algorithm was also shown to be a redundant variant of the F_4 algorithm [ACFP12].

The primary motivation to develop a new Gröbner bases algorithm from a cryptanalysis standpoint is to shift the attention from linearization-based approaches to a family of algorithms with a well-established theory. There are several proposals of Gröbner bases algorithms dedicated for cryptanalysis purposes [FJ03, JV11]. However, their exposure are limited to high-level descriptions without much emphasis on the implementation aspects. This in contrast to the fact that the efficiency of Gröbner bases algorithms are often shown by their ability to solve computational problems in practice. For instance, the efficiency of F_4 [Fau99] was demonstrated by its implementation that solved the Cyclic

9-roots problem.* The paper, however, mentioned almost no direction on how to implement the algorithm. Moreover, two well-known implementations of the Gröbner bases algorithm [Fau10, BCP97] are provided as closed-source binaries. This thesis tries to address this problem: we develop a Gröbner bases algorithm for cryptanalysis purposes and present its high-level description together with its implementation in a unified theoretical framework.

We choose to focus on the cryptanalysis of MPKCs because of one main reason: the growing interest in their development as candidates for post-quantum cryptography. In December 2016, the US National Institute for Standards and Technology (NIST) has initiated a call for proposals for a post-quantum cryptography standardization process. The goal is to develop classical cryptographic schemes that are compatible with existing network/communication protocols and considered secure against attacks by both quantum and classical computers. The security of several submitted proposals are based on the hardness of the MQ problem over finite fields. These proposals are DME [Lue17], RAINBOW [DS05], LUOV [BP17], DUALMODEMS [FPR17a], GEMSS [FPR+17b], GUI [DCP+17], HIMQ-3 [SPK17] and MQDSS [CHR+17]. Among those proposals, RAINBOW was the one that advanced the furthest and emerged as one of the Round 3 finalists [NIS20]. However, in Crypto 2022 [Beu22] Beullens introduced a new attack on RAINBOW that reduces the security level for all the proposed parameters and was mounted in practice to recover the private key for the smallest parameter. Nonetheless, this does not eliminate the interest of having MPKC schemes in the NIST post-quantum standardization mainly due to their features of having short signature and fast verification. Both features are explicitly stated as the interests of NIST in its additional call for post-quantum digital signature schemes [NIS22] where 10 out of 40 submissions are classified under the category of multivariate signatures: 3WISE [Rod23a], DME SIGN [Ln23], HPPC [Rod23b], MAYO [BCC+23], PROV [GCF+23], QR-UOV [FIH+23], SNOVA [WCD+23], TUOV [DGG+23], UOV [BCD+23], and VOX [PCF+23].

In Section 2.2 we describe that the public-key of encryption or signature schemes based on the hardness of MQ problem consists of multivariate quadratic polynomials over a finite field. Given a ciphertext (in the case of encryption schemes) or a signature (in the case of digital signature schemes), the direct approach to perform a plaintext-recovery or signature forgery attack on MQ-based public-key and digital signature schemes is to compute a Gröbner basis of the ideal generated by the polynomials in the public key. In order to gain a better confidence in their security, it is crucial to understand the complexity of computing Gröbner bases in practice. Following the same motivation,

* Cyclic n-roots problem is the problem of finding a solution for the system of equations

$$\begin{aligned} &x_1 + x_2 + \cdots + x_n = 0,\\ &x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n + x_nx_1 = 0,\\ &x_1x_2x_3 + x_2x_3x_4 + \cdots + x_nx_1x_2 = 0,\\ &\vdots\\ &x_1x_2\cdots x_n - 1 = 0. \end{aligned}$$

This is a well-known benchmark problem in computer algebra, popularized by James Davenport [Dav87].

Yasuda *et al.* introduced an open challenge to solve concrete instances of the MQ problem [YDH+15a]. The main goal of the challenge is to understand the relation among the order of a finite field, the number of variables, and the number of equations with the hardness of solving MQ problem. The result of solved challenges may contribute in determining appropriate parameters for MQ-based public-key or digital signature schemes.

Gröbner bases and Gröbner bases algorithms are also known to have applications in other areas such as algebraic geometry [AL94, Section 2.5], coding theory [BP09, Sak98], graph-coloring [Bay82, Section 3.2], integer programming [MMR91], geometric theorem proving [Wan98], etc. In this thesis we also explore other possible new applications of Gröbner bases, particularly on problems that can be expressed as a *constraint satisfaction problem* [RN10, Chapter 6] and also in the cryptanalysis of symmetric-key primitives.

1.3.2 Technical Contributions

The contribution of this thesis is divided into two parts. The first contribution deals with the algorithmic and the implementation aspect of the Gröbner bases algorithm. The second contribution proposes several new applications of Gröbner bases in different domains. All these contributions are described in Chapter 5, Chapter 6, Chapter 7, and Chapter 8.

Part of the results in Chapter 5 and Chapter 6 have been published in the following conference proceedings

> Rusydi H. Makarim, Marc Stevens. *M4GB: An Efficient Gröbner-Basis Algorithm*. Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017 [MS17].

Chapter 5 extends the results of [MS17] by providing a detailed rationale of the reduction strategy of the M4GB algorithm as an improvement over the reduction strategy of the F_4 algorithm [Fau99] (see Section 5.1). The same chapter also provides an example to better illustrate the M4GB reduction and its generalization that applies on term-polynomial multiplication (see Section 5.2). Chapter 6 also extends the results of [MS17] by giving an overview of the MQ challenge [YDH+15a] and the hybrid approach [BFP09, BFP12] in a greater details (see Section 6.1, Section 6.2 and Section 6.3). Moreover, Chapter 6 also provides cost estimations, in terms of CPU time and memory, of using the current implementation of the M4GB algorithm to solve other instances of the MQ challenge with higher parameters (see Section 6.5).

Part of the results in Chapter 7 have been published in the following journal

> Putranto H. Utomo, Rusydi H. Makarim. *Solving a Binary Puzzle*. Mathematics in Computer Science, Volume 11, Numbers 3-4 [UM17].

Chapter 7 extends the result of [UM17] in threefold. Firstly, it provides a new way to express the polynomial equation for one of the constraint of binary puzzles as a linear combination of the elementary symmetric polynomials over $\mathbb{F}_2$ (see Theorem 7.14 in Section 7.5). Secondly, it introduces a strategy to reduce the degree of the polynomials of binary puzzles over $\mathbb{F}_2$ at the expense of working over the rational field or the integer ring (see Section 7.6). Lastly, it provides a benchmark that compares the performance of the implementation of Gröbner

bases algorithms in Magma[BCP97], Singular[DGPS18], and PolyBori[BD09a] to solve systems of polynomial equations of binary puzzles over $\mathbb{F}_2$, the rational field, and the integer ring (see Section 7.7).

M4GB Gröbner Bases Algorithm

In Chapter 5, we propose a new variant of the Gröbner bases algorithm called M4GB algorithm. The algorithm is designed to consistently operate on and maintain polynomials in tail-reduced form with respect to the current intermediate basis. In contrast to the F_4 algorithm that computes the (reduced)-row echelon form of coefficient matrices that includes reducible terms in the tail of the corresponding polynomials, the strategy of M4GB can be seen as a way to perform similar computation on coefficient matrices that contain only non-reducible terms in the tail of each polynomial. We also introduce a new recursive reduction algorithm on polynomials of the form $\mathtt{t}f$ where $\mathtt{t}$ and f are a term and a polynomial in $\mathbb{F}[x_1, \ldots, x_n]$ respectively. The M4GB algorithm is described in a way that unifies high-level description together with its implementation aspects. Although this is different from the usual tradition to describe a Gröbner bases algorithm, our goal here is to present it without obscuring the nature of its implementation.

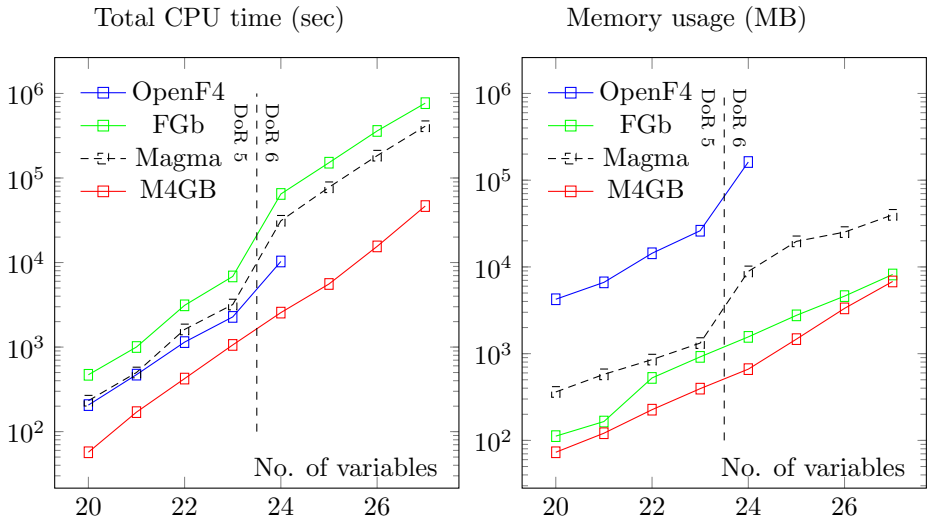


Figure 1: *The CPU time and memory comparison of* M4GB *with other Gröbner bases implementation for system of equations with* $2n$ *equations over* $\mathbb{F}_{31}$*, where* n *is the number of variables. The vertical dashed line separates the parameter with different degree of regularity (DoR).*

Following a similar approach as in [Fau99, Fau02], we demonstrate the efficiency of the M4GB algorithm by implementing it in practice and comparing its performance against other existing implementations of the Gröbner bases algorithms. The implementation is available in the following link

`https://github.com/cr-marcstevens/m4gb`.

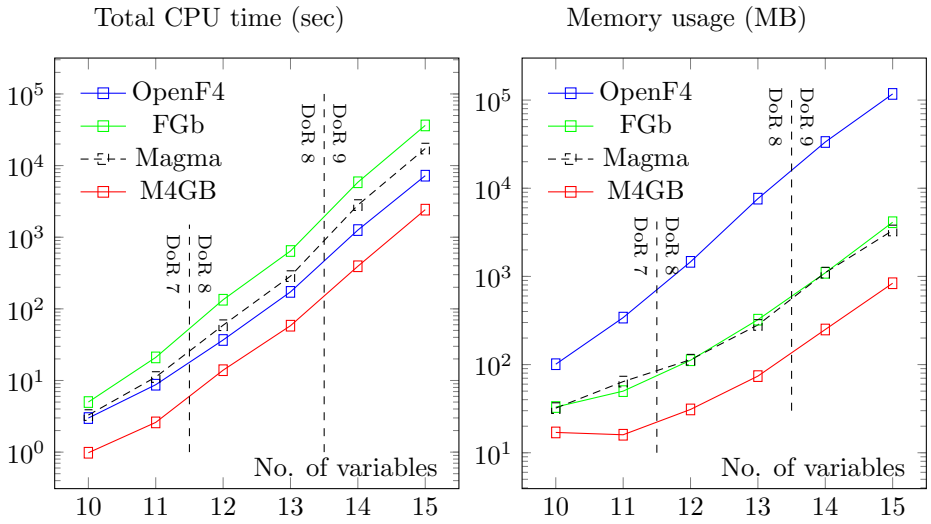


Figure 2: *The CPU time and memory comparison of* M4GB *with other Gröbner bases implementation for system of equations with* $n+1$ *equations over* $\mathbb{F}_{31}$*, where* n *is the number of variables. The vertical dashed line separates the parameter with different degree of regularity (DoR).*

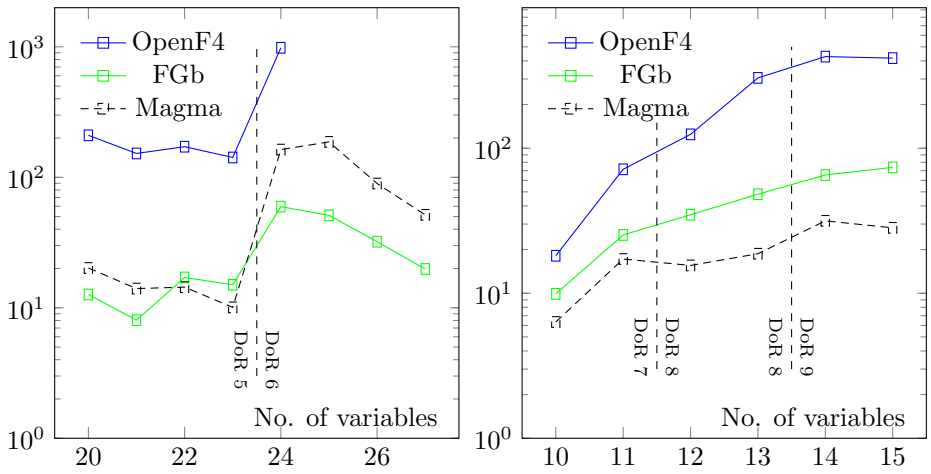


Figure 3: *The ratio of time-memory product of Magma, FGb, and OpenF4 relative to the time-memory product of* M4GB*. The left graph is the result on systems with* n *variables and* $2n$ *equations whereas the right one is the result on systems with* n *variables and* $n+1$ *equations. The vertical dashed line separates the parameter with different degree of regularity (DoR).*

We compare the implementation of our open-source M4GB with three state-of-the-art Gröbner bases implementations, namely Magma [BCP97], FGb [Fau10], and OpenF4 [CVJ]. We compare the performance and memory usage of our M4GB algorithm against existing Gröbner bases implementations for system of polynomial equations that represent a public-key of MPKC schemes, i.e., overdefined quadratic systems of equations over finite fields. The results of the benchmarks are given in Figure 1 and Figure 2. Although OpenF4, FGb, and Magma appear to provide a trade-off between CPU time and memory usage, our implementation of M4GB consistently has the best performance and the least memory usage in all benchmark parameters. For the case of $2n$ equations with n variables, M4GB performs 2.5 up to 4 times faster than OpenF4 (the second fastest implementation). M4GB also has the least memory usage by a factor between 1.2 and 2.35 compared to FGb (having the second smallest memory usage). For systems with $n+1$ equations and n variables, M4GB also emerges as the most efficient algorithm to solve such systems. It performs 2.6 up to 3.3 times faster than OpenF4 as the second fastest implementation and it uses 1.9 up to 4.4 times less memory compared to FGb and Magma.

The efficiency of the M4GB algorithm also contributes to the cryptanalysis of existing multivariate-based post-quantum digital signatures. Recently Beullens [Beu24] utilized our implementation of the M4GB algorithm to demonstrate a practical attack against SNOVA [WCD+23], one of the candidate of the NIST on-ramp post-quantum signature. The attack itself consists of multiple steps where the M4GB algorithm is used in the last one, which is also the bottleneck of the attack, that requires solving multiple systems of 14 polynomal equations in 13 variables over $\mathbb{F}_{16}$ (see page 13-15 of [Beu24]).

MQ Challenge Computational Records

Chapter 6 presents the application of M4GB as a Gröbner bases algorithm to solve some concrete challenges of MQ problems over finite fields. Particularly, M4GB set a record in the *Fukuoka MQ Challenge* for parameters that represent the public-key in MQ-based digital signature algorithms.* A short summary of challenges solved by M4GB is available in Table 1. Moreover, in the same chapter we provide an estimation of CPU time and memory to solve larger parameters of type V and VI of MQ challenge. The result is described in Table 2.

Solving Binary Puzzles using Gröbner Bases

Chapter 7 presents an application of Gröbner bases algorithms to find solutions for *binary puzzles*. A binary puzzle is a Sudoku-like puzzle with an entry in each cell taken from the set $\{0,1\}$ instead of $\{0,1,\ldots,9\}$. Let $n \geq 4$ be an even integer and $[n] = \{1,2,\ldots,n\}$. We define the *initial configuration of a binary puzzle* as a subset $M \subseteq [n]^2 \times \{0,1\}$. The set M tells that the value at row i and column j of the binary puzzle is equal to $v_{i,j}$, for all $((i,j),v_{i,j}) \in M$. An $n \times n$ binary puzzle with the initial configuration $M \subseteq [n]^2 \times \{0,1\}$ is solvable if there exists an assignment for each cell that satisfies the following constraints :

* The challenges are random instances of the MQ problem over finite fields that represent MQ-based public-key cryptography and digital signature. All challenges and existing records are available at `https://www.mqchallenge.org`.

Type	(n, m)	k	Duration	Search Space Covered
V	$(24, 16)$	1	$\approx$ 9.3 hours	29%
V	$(25, 17)$	1	$\approx$ 46.33 hours	46.5%
V	$(27, 18)$	1	$\approx$ 10.9 days	95.7%
V	$(28, 19)$	1	$\approx$ 69.75 days	71.5%
VI	$(24, 16)$	1	$\approx$ 1.2 hours	22.6%
VI	$(25, 17)$	1	$\approx$ 9.87 hours	64.5%
VI	$(27, 18)$	1	$\approx$ 31.48 hours	12.9%
VI	$(28, 19)$	1	$\approx$ 7.61 days	54.8%
VI	$(30, 20)$	2	$\approx$ 11.32 days	21.5%

Table 1: *Summary of solved MQ challenges with n variables and m equations using the implementation of* M4GB. *Type V and VI consist of polynomials defined over $\mathbb{F}_2$s and $\mathbb{F}_{31}$ respectively. For each challenge, we solved q^k subsystems by substituting k variables with all possible values in $\mathbb{F}_q$, where q is the order of the finite field. The subsystems are generated after fixing the value of $n - m$ chosen variables.*

m	$m-n$		$\approx$ CPU hours		$\approx$ Memory (GB)	
	V	VI	V	VI	V	VI
20	1	-	402045	-	210	-
21	1	3	$2.85 \cdot 10^6$	217452	764	5.47
22	1	2	$2.02 \cdot 10^7$	$1.39 \cdot 10^6$	2776	189
23	1	2	$1.44 \cdot 10^8$	$8.94 \cdot 10^6$	10087	654
24	1	3	$1.02 \cdot 10^9$	$1.43 \cdot 10^7$	36643	130
25	1	3	$7.23 \cdot 10^9$	$1.57 \cdot 10^8$	133110	528
26	1	3	$5.13 \cdot 10^{10}$	$8.15 \cdot 10^8$	483536	1631
27	1	3	$3.64 \cdot 10^{11}$	$4.24 \cdot 10^9$	$1.76 \cdot 10^6$	5042
28	1	3	$2.58 \cdot 10^{12}$	$2.20 \cdot 10^{10}$	$6.40 \cdot 10^6$	15584
29	1	3	$1.83 \cdot 10^{13}$	$1.15 \cdot 10^{11}$	$2.32 \cdot 10^7$	$4.81 \cdot 10^4$
30	1	3	$1.30 \cdot 10^{14}$	$5.95 \cdot 10^{11}$	$8.44 \cdot 10^7$	$1.49 \cdot 10^5$
31	1	3	$9.23 \cdot 10^{14}$	$3.10 \cdot 10^{12}$	$3.06 \cdot 10^8$	$4.61 \cdot 10^5$
32	1	3	$6.55 \cdot 10^{15}$	$1.61 \cdot 10^{13}$	$1.11 \cdot 10^9$	$1.42 \cdot 10^6$
33	1	3	$4.64 \cdot 10^{16}$	$8.37 \cdot 10^{13}$	$4.04 \cdot 10^9$	$4.40 \cdot 10^6$
34	1	3	$3.30 \cdot 10^{17}$	$4.35 \cdot 10^{14}$	$1.47 \cdot 10^{10}$	$1.36 \cdot 10^7$
35	1	5	$2.34 \cdot 10^{18}$	$1.95 \cdot 10^{15}$	$5.32 \cdot 10^{10}$	$8.37 \cdot 10^4$

Table 2: *Estimated cost to solve larger parameters of type V (top) and VI (bottom) using our implementation of* M4GB *algorithm running on Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30 GHz. Memory usage is for a single subsystem and the total system memory required depends on the number of subsystems being solved simultaneously.*

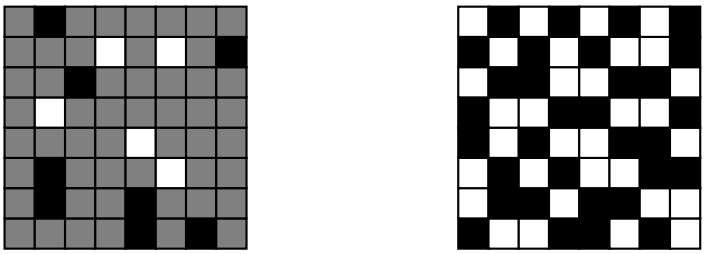


Figure 4: *An example of initial configuration of an 8×8 binary puzzle (left) and its corresponding solution (right). Note that multiple solutions may exist depending on the size and the initial configuration.*

1. There exists no three (3) consecutive cells, both vertically and horizontally, which are filled with the same value
2. The number of zeros and ones in each row and column must be equal, i.e., their number in each row and column must be equal to $n/2$
3. Every two distinct rows (resp. columns) must not be equal.

Solving a binary puzzle can then be viewed as a *constraint satisfaction problem*. Our approach is to see each constraint that must be satisfied as a polynomial equation. Since each entry in the puzzle is taken from the set $\{0, 1\}$, the first natural approach is to look at each constraint as a polynomial over $\mathbb{F}_2$.

Theorem 1.1. *An $n \times n$ Binary puzzle with the initial configuration M is solvable if and only if the following system of polynomial equations over $\mathbb{F}_2$ has a solution in $\mathbb{F}_2^{(n^2)}$:*

$$
\begin{aligned}
& x_{i,j+1}x_{i,j+2} + x_{i,j}x_{i,j+2} + x_{i,j+2} \\
& \qquad + x_{i,j}x_{i,j+1} + x_{i,j+1} + x_{i,j} + 1, && 1 \le i \le n, 1 \le j \le n-2 \\
& x_{i+1,j}x_{i+2,j} + x_{i,j}x_{i+2,j} + x_{i+2,j} \\
& \qquad + x_{i,j}x_{i+1,j} + x_{i+1,j} + x_{i,j} + 1, && 1 \le j \le n, 1 \le i \le n-2 \\
& 1 + \sum_{\substack{k=1 \\ n/2 \preceq k}}^{n} \sigma_{k,n}(x_{i,1}, \ldots, x_{i,n}), && 1 \le i \le n \\
& 1 + \sum_{\substack{k=1 \\ n/2 \preceq k}}^{n} \sigma_{k,n}(x_{1,j}, \ldots, x_{n,j}), && 1 \le j \le n \\
& \prod_{k=1}^{n} (x_{i,k} + x_{j,k} + 1), && 1 \le i \le n-1, i+1 \le j \le n \\
& \prod_{k=1}^{n} (x_{k,i} + x_{k,j} + 1), && 1 \le i \le n-1, i+1 \le j \le n \\
& x_{i,j}^2 + x_{i,j} && 1 \le i \le n, 1 \le j \le n. \\
& x_{i,j} + v_{i,j} && \forall ((i,j), v_{i,j}) \in M
\end{aligned}
$$

where $\sigma_{k,n}(x_1, \ldots, x_n)$ denotes the k-th elementary symmetric polynomial in n-variables $x_1, \ldots, x_n$ and $a \preceq b$, $(a, b \in \mathbb{Z}_{\ge 0})$ if the binary representation of a is covered by the binary representation of b, .

The second approach is to look at each constraint as polynomial over $\mathbb{Q}$ or $\mathbb{Z}$. One advantage of this approach is the polynomial representation for the second

constraint is linear, thus significantly reducing the degree of the same constraint at the expense of defining the polynomials over $\mathbb{Q}$ or $\mathbb{Z}$.

Theorem 1.2. *An $n \times n$ binary puzzle with the initial configuration M is solvable if and only if the following system of polynomial equations over $\mathbb{Q}$ (or $\mathbb{Z}$) has a solution in $\mathbb{Q}^{(n^2)}$ (or $\mathbb{Z}^{(n^2)}$):*

$$\begin{aligned}
&x_{i,j+1}x_{i,j+2} + x_{i,j}x_{i,j+2} - x_{i,j+2} && \\
&\qquad +x_{i,j}x_{i,j+1} - x_{i,j+1} - x_{i,j} + 1, && 1 \le i \le n, 1 \le j \le n-2 \\
&x_{i+1,j}x_{i+2,j} + x_{i,j}x_{i+2,j} - x_{i+2,j} && \\
&\qquad +x_{i,j}x_{i+1,j} - x_{i+1,j} - x_{i,j} + 1, && 1 \le j \le n, 1 \le i \le n-2 \\
&\sum_{j=1}^{n} x_{i,j} - n/2, && 1 \le i \le n \\
&\sum_{i=1}^{n} x_{i,j} - n/2, && 1 \le j \le n \\
&\prod_{k=1}^{n} (x_{i,k} + x_{j,k} - 1), && 1 \le i \le n-1, i+1 \le j \le n \\
&\prod_{k=1}^{n} (x_{k,i} + x_{k,j} - 1), && 1 \le i \le n-1, i+1 \le j \le n \\
&x_{i,j}^2 - x_{i,j} && 1 \le i \le n, 1 \le j \le n \\
&x_{i,j} - v_{i,j} && \forall((i,j), v_{i,j}) \in M.
\end{aligned}$$

The solutions for a binary puzzle can then be obtained by computing a Gröbner bases of the ideal generated by either the polynomials in Theorem 1.1 or Theorem 1.2.

In order to understand the impact of reducing the degree of some equations at the expense of changing the ring where coefficients of the polynomials are defined, we also compare the performance of Gröbner bases algorithm in practice to solve system of equations that represent binary puzzles over $\mathbb{F}_2$, $\mathbb{Q}$, and $\mathbb{Z}$.

New Characterizations of S-Boxes Criteria

Chapter 8 introduces a new-point-of-view to look at cryptographic properties of a vectorial Boolean functions from the settings of ideals in multivariate polynomial ring. We propose new characterization for notions related to the nonlinearity, differential, and autocorrelation from ideal-theoretic viewpoint. The primary contributions are summarized in the following theorems.

Theorem 1.3. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and $\mathcal{R}$ be a polynomial ring over $\mathbb{F}_2$ in $n+m$ variables. For any $\boldsymbol{a} \in \mathbb{F}_2^n$ and $\boldsymbol{b} \in \mathbb{F}_2^m$ there exists an ideal I of $\mathcal{R}$ such that the bias of a linear approximation on S with input mask $\boldsymbol{a}$ and output mask $\boldsymbol{b}$ is equal to*

$$2^{-n} \cdot (\dim_{\mathbb{F}_2}(\mathcal{R}/I) - 2^{n-1}).$$

Theorem 1.4. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and $\mathcal{R}$ be a polynomial ring over $\mathbb{F}_2$ in $2(n+m)$ variables. For any $\boldsymbol{\alpha} \in \mathbb{F}_2^n, \boldsymbol{\beta} \in \mathbb{F}_2^m$ there exists an ideal I of $\mathcal{R}$ such that the probability of a differential $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is equal to*

$$2^{-n} \cdot \dim_{\mathbb{F}_2}(\mathcal{R}/I).$$

Theorem 1.5. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and $\mathcal{R}$ be a polynomial ring over $\mathbb{F}_2$ in $2(n+m)$ variables. For any $\boldsymbol{\alpha} \in \mathbb{F}_2^n, \boldsymbol{b} \in \mathbb{F}_2^m$ there exists an ideal I of $\mathcal{R}$ such that the autocorrelation of the component function $\boldsymbol{b} \cdot S(x)$ at $\boldsymbol{\alpha}$ is equal to*

$$2^n - 2 \cdot \dim_{\mathbb{F}_2}(\mathcal{R}/I).$$

In the same chapter we also exhibit new approaches to compute the bias of a linear approximation, the probability of a differential, and the autocorrelation of a component function that utilize Gröbner bases algorithms. To the best of our knowledge, this is the first work that establishes connections among cryptographic properties of vectorial Boolean functions and computational commutative algebra.

1.3.3 Thesis Outline

The remainder of this thesis consists of seven chapters.

Chapter 2 is a survey on various aspects of MQ problems and solving systems of polynomial equations in cryptology. It describes the intractability of the MQ problem and how it can be used to construct public-key and digital signature schemes. The chapter also describes the relevance of solving MQ problem and systems of polynomial equations in the cryptanalysis of symmetric-key primitives. It also provides a brief survey of existing techniques in the literatures to solve systems of polynomial equations.

Chapter 3 discusses the Gröbner bases and their applications. It serves as the preliminary chapter where we explain all notations and terminologies used throughout this thesis. In the same chapter we also describe two applications of Gröbner bases that are relevant within the scope of this thesis: solving the ideal membership problem and solving systems of multivariate polynomial equations.

Chapter 4 focuses on the Gröbner bases algorithms. We shall explain Buchberger [Buc65, Buc06] and the F_4 algorithm [Fau99] in Section 4.1 and Section 4.2 respectively. In the same chapter, we also discuss several improvement strategies for both algorithms, where their main goal is to detect unnecessary computations in advance. Moreover, we also describe the XL algorithm [CKPS00] in Section 4.3. Even though it was not originally proposed to compute Gröbner bases of an ideal, it can actually be viewed as a redundant variant of the F_4 algorithm.

Chapter 5 explains the M4GB algorithm. It covers the description of the algorithm, the reduction strategy, and the rationale behind its design. The performance comparison of M4GB against FGb [Fau10], Magma [BCP97], and OpenF4 [CVJ] are given in Section 5.7.

Chapter 6 describes the application of the M4GB algorithm to solve concrete challenges of MQ problems proposed in [YDH$^+$15a, YDH$^+$15b] for parameters that represent signature-type MPKC, i.e., an underdefined system of equations over a finite field. We give an overview of existing strategies to solve such a system of equations in Section 6.3. We also discuss the cost of solving larger parameters of signature-type MQ challenge in Section 6.5.

Chapter 7 explains the application of the Gröbner bases algorithm to solve binary puzzles. In particular, the proof of Theorem 1.1 and Theorem 1.2 are given in Section 7.5 and Section 7.6 respectively. We also compare the performance of different implementations of Gröbner bases algorithms to solve sys-

tems of polynomial equations over $\mathbb{F}_2$, $\mathbb{Q}$, and $\mathbb{Z}$ that represent a binary puzzle in Section 7.7.

Chapter 8 presents the relation of cryptographic properties of vectorial Boolean functions and their corresponding ideals. The proof of Theorem 1.3, Theorem 1.4, and Theorem 1.5 are given in Section 8.2, Section 8.3, and Section 8.4 respectively. In each of those sections we also present the algorithms based on the Gröbner bases that compute the bias of a linear approximation, the probability of a differential and the autocorrelation. In relation with autocorrelation, we also discuss a possible way to look at the existence of a linear structure as ideal membership problem in Subsection 8.4.1.

1.3.4 Publications

In addition to [MS17, UM17], the author has co-authored the following publication related to the study of the intractability of the MQ problem and multivariate public-key cryptosystems. The results, though related, are not covered in this dissertation.

[BMSV22] Emanuele Bellini, Rusydi H. Makarim, Carlo Sanna, Javier A. Verbel. *An Estimator for the Hardness of the MQ Problem*. 13th International Conference on Cryptology in Africa (AFRICACRYPT), Fez, Morocco, 2022, pp. 323-347.

Moreover, the author has also contributed to the following publications in the area of cryptanalysis of symmetric-key primitives. Hence, these results are not covered in this dissertation.

[HDRM23] Solane El Hirch, Joan Daemen, Raghvendra Rohit, Rusydi H. Makarim. *Twin Column Parity Mixers and Gaston - A New Mixing Layer and Permutation*. 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part III, pp. 475-506.

[BGG+23b] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, Thomas Peyrin. *Boosting Differential-Linear Cryptanalysis of ChaCha7 with MILP*. (2023). IACR Transactions on Symmetric Cryptology, Volume 2023, Issue 2, pp. 189-223.

[BGG+23a] Emanuele Bellini, David Gérault, Juan Grados, Yun Ju Huang, Rusydi H. Makarim, Mohamed Rachidi, Sharwan K. Tiwari. *CLAASP: a Cryptographic Library for the Automated Analysis of Symmetric Primitives*. Selected Areas in Cryptography (SAC), Fredericton, New Brunswick, Canada, 2023, pp. 387-408.

[BGG+23c] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, Thomas Peyrin. *Fully Automated Differential-Linear Attacks Against ARX Ciphers*. Cryptographers' Track at the RSA Conference (CT-RSA), San Francisco, USA, 2023, pp 252–276.

[MR22] Rusydi H. Makarim and Raghvendra Rohit. *Towards Tight Differential Bounds of Ascon: A Hybrid Usage of SMT and MILP*. (2022). IACR Transactions on Symmetric Cryptology, Volume 2022, Issue 3, pp. 303-340.

[BPM+21] Emanuele Bellini, Alessandro De Piccoli, Rusydi H. Makarim, Sergio Polese, Lorenzo Riva, Andrea Visconti. *New Records of Pre-image Search of Reduced SHA-1 Using SAT Solvers*. Proceedings of the Seventh International Conference on Mathematics and Computing (ICMC), Shibpur, India, 2021, pp. 141-151.

[BGMS24] Emanuele Bellini, Juan Grados, Rusydi H. Makarim, Carlo Sanna. *Finding Differential Trails on ChaCha by Means of State Functions*. International Journal of Applied Cryptography, Volume 4, No. 3/4, 2024, pp. 156-175.

[BBM23] Stefano Barbero, Emanuele Bellini, Rusydi H. Makarim. *Rotational Analysis of ChaCha Permutation*. Advances in Mathematics of Communications, Volume 17, Issue 6, 2023, pp. 1422-1439.

[BM22] Emanule Bellini, Rusydi H. Makarim.*Functional Cryptanalysis: Application to reduced-round Xoodoo*. In IACR Cryptology ePrint Archive (2022). IACR ePrint: 2022/134.

[MT14] Rusydi H. Makarim and Cihangir Tezcan. *Relating Undisturbed Bits to Other Properties of Substitution Boxes*. Lightweight Cryptography for Security and Privacy - Third International Workshop (LightSec), Istanbul, Türkiye, 2014, pp. 109-125.

Finally, the author has also contributed to the following publications related to the implementation of code-based post-quantum cryptography. Hence, these results are also excluded in this dissertation.

[BCM+19] Emanuele Bellini, Florian Caullery, Rusydi H. Makarim, Marc Manzano, Chiara Marcolla, Victor Mateu. *Advances and Challenges of Rank Metric Cryptography Implementations*. IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, 2019, pp. 325-328.

[MAB+21] Carlos Aguilar Melchor, Nicolas Aragon, Emanuele Bellini, Florian Caullery, Rusydi H. Makarim, Chiara Marcolla. *Constant Time Algorithms for ROLLO-I-128*. SN Computer Science, Volume 2, Number 5, September 2021, article number 382.

Part II
Technical Preliminaries

2 Multivariate Quadratic (MQ) Problems in Cryptology

Contents

Algebraic cryptanalysis is a generic technique that allows assessment of numerous cryptographic schemes. Its main principle is to express a cryptanalytic problem (e.g., key-recovery, plaintext-recovery, etc) into a system of multivariate polynomial equations. The public parameters, such as description of encryption schemes or public-keys, are used to express the system of equations. The construction of the equations are done in a way that allows a correspondence between the solutions of the system with the secret component of the cryptographic primitive. The polynomials are, in general, quadratic and the coefficients are taken from a finite field.

The security level of an encryption scheme against algebraic cryptanalysis is linked with the difficulty of solving system of polynomial equations over a finite field. It is often the case that a cryptographic primitive can be represented by different system of equations. This cryptanalytic technique has been successfully applied to break the security of multivariate public-key/signature schemes [FJ03, FLP08] as well as certain families of stream ciphers [CP02, CM03, Cou03].

This chapter is meant to provide a bird's-eye view on various aspects of algebraic cryptanalysis. We begin by describing a computationally intractable problem related to solving a system of quadratic polynomial equations over a field called the *Multivariate Quadratic* (MQ) problem in Section 2.1. This hard computational problem is also used as a foundation to construct post-quantum cryptographic primitives. The description of public-key cryptography based on the difficulty of solving polynomial equations is given in Section 2.2. This will be followed by Section 2.3 which provides a survey on how algebraic cryptanalysis plays a role in the cryptanalysis of symmetric-key primitives. We also give a brief overview of the algebraic cryptanalysis against post-quantum

cryptographic schemes in Section 2.4. We will then discuss several methods that have been applied to find a solution of multivariate polynomial equations from different cryptographic primitives. Some of the methods mentioned are algebraic in nature while others require reductions of solving multivariate polynomial equations to other intractable problems. They will be explained in Section 2.5.

2.1 The Intractability of the MQ Problem

The central subject of this thesis is techniques to solve quadratic systems of polynomial equations over a finite field. The term "solve" or "solving" can be interpreted in multiple ways. One may view it as an effort to recover all solutions of the system of equations, including the one in the algebraic closure of the coefficient field. However, in the context of cryptanalysis the term "solve" and "solving" are restricted to finding a solution that lies in the coefficient field. Definition 2.1 formally defines this perspective.

Definition 2.1 (Polynomial System Solving)**.** *Let R be a commutative ring with a multiplicative identity $1 \neq 0$ and let $f_1, \ldots, f_m \in R[x_1, \ldots, x_n]$ be m polynomials over n-variables $x_1, \ldots, x_n$ with coefficients in R. The polynomial system solving problem (PoSSo) for $f_1, \ldots, f_m$ over R is the problem of finding an element $(a_1, \ldots, a_n) \in R^n$ such that $f_i(a_1, \ldots, a_n) = 0$ for all $i \in \{1, \ldots, m\}$. We call the set $\{f_1, \ldots, f_m\}$ an instance of the PoSSo problem over R.*

Definition 2.2 (MQ problem)**.** *An instance $F = \{f_1, \ldots, f_m\} \subset R[x_1, \ldots, x_n]$ of the PoSSo problem over R is called a multivariate quadratic (MQ) problem over R if the degree of f_i, $1 \leq i \leq m$, is (at most) quadratic.* The decisional MQ problem over R asks whether there exists an assignment $(a_1, \ldots, a_n) \in R^n$ such that $f_i(a_1, \ldots, a_n) = 0$ for all $i \in \{1, \ldots, m\}$.*

We will show that the decisional MQ problem over $\mathbb{F}_2$ is NP-complete. The proof is based on the reduction to the following Boolean satisfiability problem.

Definition 2.3 (3-CNF SAT Problem)**.** *Let $X = \{X_1, \ldots, X_n\}$ be a set of Boolean variables, let $L = \{X_1, \neg X_1, \ldots, X_n, \neg X_n\}$ be the set of its corresponding literals (where $\neg X_i$ denotes the negation of X_i), and let $\wedge, \vee$ denote Boolean* `AND` *and* `OR` *respectively. A Boolean formula $C = \{c_1, \ldots, c_m\}$ is in 3-CNF (Conjunctive Normal Form) if it is of the form*

$$c_1 \wedge c_2 \wedge \cdots \wedge c_m$$

where $c_i \in (L \cup L^2 \cup L^3)$ is a Boolean `OR` *of at most three literals (we shall call c_i a clause). The corresponding 3-CNF SAT problem is the problem to determine if there is a valid assignment $A \in \{\texttt{True}, \texttt{False}\}^n$ for X such that all $c_i \in C$ are true and, hence, C itself is satisfied.*

The Cook-Levin theorem [Coo71, Lev73] states that the Boolean satisfiability problem is NP-complete. Based on this result, the following proposition shows that the decisional MQ problem over $\mathbb{F}_2$ is also NP-complete.

Proposition 2.4. *The decisional MQ problem over $\mathbb{F}_2$ is NP-complete.*

* See Definition 3.12 for the definition of the degree of a polynomial.

Proof. The proof here is adapted from subsection 2.5.1 of [Wol05]. The outline of this proof consists of two parts. We first show that solving an instance of MQ problem over $\mathbb{F}_2$ is in NP, i.e., there exists a non-deterministic polynomial-time algorithm that finds a solution if one exists. This will be followed by the second part that shows there exists a polynomial-time reduction algorithm that transforms an instance of 3-CNF SAT problem to an instance of MQ problem over $\mathbb{F}_2$.

Let $(f_1, \ldots, f_m) \in \mathbb{F}_2[x_1, \ldots, x_n]$ be an instance of MQ problem over $\mathbb{F}_2$. There exists a non-deterministic algorithm that finds, in polynomial time, an element $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ such that $f_i(a_1, \ldots, a_n) = 0$ for all $i = 1, 2, \ldots, m$ if one exists. The algorithm is described as follows

1. Select an element $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$.
2. If $f_i(a_1, \ldots, a_n) = 0$ for all $i = 1, 2, \ldots, m$, then return `True`. Otherwise, go to step 1.

Note that the complexity of step (2) requires $m \sum_{i=0}^{2} \binom{n}{i} = m(1+n+n(n-1)/2)$ field operations, thus step (2) has polynomial-time complexity. The algorithm is terminated if step (2) outputs `True`. Otherwise, it goes to an infinite loop. Thus, the MQ problem over $\mathbb{F}_2$ is in NP.

In order to prove the NP-completeness, we reduce an instance of 3-CNF SAT problem to an instance of MQ problem over $\mathbb{F}_2$. Let C be a set of clauses of n Boolean variables in 3-CNF and $|C| = m$. Here we associate the Boolean value `True` to $1 \in \mathbb{F}_2$ and `False` to $0 \in \mathbb{F}_2$. Similarly, for each $i = 1, \ldots, n$ we associate each Boolean variable X_i to its corresponding indeterminate x_i in $\mathbb{F}_2[x_1, \ldots, x_n]$. The following Algorithm 2.1 transforms C to a set of multivariate polynomial equations over $\mathbb{F}_2$.

Input: A set of clauses $C = \{c_1, \ldots, c_m\}$ in CNF with variables $X_1, \ldots, X_n$
Output: A subset $F = \{f_1, \ldots, f_m\} \subset \mathbb{F}_2[x_1, \ldots, x_n]$

```
1  F ← {}
2  for i = 1 to m do
3      f_i ← 1 ∈ F_2[x_1, ..., x_n]
4      forall ℓ ∈ c_i do
5          if ℓ = ¬X_j, j ∈ {1, ..., n} then
6              f_i ← f_i · x_j
7          else if ℓ = X_j, j ∈ {1, ..., n} then
8              f_i ← f_i · (x_j − 1)
9      F ← F ∪ {f_i}
10 return F
```

Algorithm 2.1: Convert a 3-CNF problem into a MQ-problem over $\mathbb{F}_2$.

Note that an assignment $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ to the variables of a polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ is considered "valid" if $f(a_1, \ldots, a_n) = 0$.* Each clause $c_i \in C$

* By associating `True` $\leftrightarrow$ 1 and `False` $\leftrightarrow$ 0, the "satisfiability" of a polynomial is the opposite of a clause. However, this does not affect the correctness of the result since the satisfiability is not related with any assignment to the variables.

is transformed to its corresponding polynomial equation $f_i \in \mathbb{F}_2[x_1, \ldots, x_n]$ of degree ≤ 3.

In order to reduce the degree of polynomials in F to quadratic, we introduce $n(n-1)/2$ additional variables $y_{i,j}$ that substitute the quadratic monomials $x_i x_j$, $1 \leq i < j \leq n$. At the same time $n(n-1)/2$ equations of the form $x_i x_j + y_{i,j}$ are also added to F. This yields a system of $m + n(n-1)/2$ quadratic polynomials in $n(n+1)/2$ variables. If there exists a solution for this system of equations, then we also have a solution to the 3-CNF SAT.* Since all the steps require polynomial time and space, we have reduced an instance of 3-CNF SAT in polynomial time to an instance of the MQ problem over $\mathbb{F}_2$. Therefore, the decisional MQ problem over $\mathbb{F}_2$ is NP-complete. ■

Proposition 2.5. *Let D be a finite integral domain. Then, the MQ problem over D is NP-hard. In particular, if the addition and multiplication in D are polynomial time operations, then the decisional MQ problem over D is NP-complete.*

Proof. The following proof is a summary of subsection 2.5.2 in [Wol05]. We first prove that MQ over a finite integral domain D is NP-hard. The proof is based on reduction of an instance of MQ over $\mathbb{F}_2$ to an instance of MQ over D.

Consider the following quadratic system of m equations in n variables over $\mathbb{F}_2$

$$\begin{aligned} \sum_{1 \leq j < k \leq n} a_{j,k}^{(1)} x_j x_k + \sum_{j=1}^{n} b_j^{(1)} x_j + c^{(1)} &= 0 \\ &\vdots \\ \sum_{1 \leq j < k \leq n} a_{j,k}^{(m)} x_j x_k + \sum_{j=1}^{n} b_j^{(m)} x_j + c^{(m)} &= 0 \end{aligned}$$

where $a_{j,k}^{(i)}, b_j^{(i)}, c^{(i)} \in \mathbb{F}_2$ are constants, for $1 \leq i \leq m$. Each polynomial above is transformed into the following system of polynomial equations over $\mathbb{F}_2$

$$\begin{aligned} a_{1,2}^{(i)} x_1 x_2 &= y_{1,2}^{(i)} \\ a_{1,3}^{(i)} x_1 x_3 &= y_{1,2}^{(i)} + y_{1,3}^{(i)} \\ &\vdots \\ a_{n-1,n}^{(i)} x_{n-1} x_n &= y_{n-2,n}^{(i)} + y_{n-1,n}^{(i)} \\ b_1^{(i)} x_1 &= y_{n-1,n}^{(i)} + z_1^{(i)} \\ b_2^{(i)} x_2 &= z_1^{(i)} + z_2^{(i)} \\ &\vdots \\ b_{n-1}^{(i)} x_{n-1} &= z_{n-2}^{(i)} + z_{n-1}^{(i)} \\ b_n^{(i)} x_n + c^{(i)} &= z_{n-1}^{(i)} \end{aligned} \tag{2.1}$$

* For instance, a clause $(X_1 \vee X_2 \vee \neg X_3)$ is converted into a polynomial $f = (x_1 + 1)(x_2 + 1)x_3$ by Algorithm 2.1. The set of solutions to f is $\mathbb{F}_2^3 \setminus \{(0,0,1)\}$ and one can also verify that the set of valid assignments to X_1, X_2, X_3 is $\{\texttt{True}, \texttt{False}\}^3 \setminus \{(\texttt{False}, \texttt{False}, \texttt{True})\}$.

where $y_{j,k}^{(i)}$ and $z_j^{(i)}$ are new variables. Each polynomial is transformed into a set of $n(n+1)/2$ equations with $n+\binom{n}{2}+(n-1) = (n(n+3)-2)/2$ variables. Thus, the original system of equations is transformed into a system of $mn(n+1)/2$ equations in $n + m\binom{n}{2} + m(n-1) = (2n + m(n-1)(n+2))/2$ variables. This step requires polynomial-time in the size of the input.

The next step is to transfer the newly constructed system of equations over $\mathbb{F}_2$ into a system of equations over D. This is done by considering the assignment on each variables $x_j, y_{j,k}^{(i)}, z_j^{(i)}$ using the elements of D and each coefficient $a_{j,k}^{(i)}, b_j^{(i)}, c^{(i)} \in \mathbb{F}_2$ is replaced by either $0 \in D$ or $1 \in D$. The multiplication in $\mathbb{F}_2$ is replaced by the multiplication in D. However, the addition operation in $\mathbb{F}_2$ needs to be replaced by $(x+y)((1+1)-(x+y))$ in D. Thus, from (2.1) we obtain the following system of equations over D

$$\begin{aligned}
a_{1,2}^{(i)} x_1 x_2 &= y_{1,2}^{(i)} \\
a_{1,3}^{(i)} x_1 x_3 &= (y_{1,2}^{(i)} + y_{1,3}^{(i)})((1+1) - (y_{1,2}^{(i)} + y_{1,3}^{(i)})) \\
&\vdots \\
a_{n-1,n}^{(i)} x_{n-1} x_n &= (y_{n-2,n}^{(i)} + y_{n-1,n}^{(i)})((1+1) - (y_{n-2,n}^{(i)} + y_{n-1,n}^{(i)})) \\
b_1^{(i)} x_1 &= (y_{n-1,n}^{(i)} + z_1^{(i)})((1+1) - (y_{n-1,n}^{(i)} + z_1^{(i)})) \\
b_2^{(i)} x_2 &= (z_1^{(i)} + z_2^{(i)})((1+1) - (z_1^{(i)} + z_2^{(i)})) \\
&\vdots \\
b_{n-1}^{(i)} x_{n-1} &= (z_{n-2}^{(i)} + z_{n-1}^{(i)})((1+1) - (z_{n-2}^{(i)} + z_{n-1}^{(i)})) \\
z_{n-1}^{(i)} &= (b_n^{(i)} + c^{(i)})((1+1) - (b_n^{(i)} + c^{(i)}))
\end{aligned}$$

for all $i = 1, \ldots, m$. In order to restrict the solution in the set $\{0,1\} \subset D$ we add new equations of the form $x(1-x)$ for each variable. The number of such equation is equal to $(2n + m(n-1)(n+2))/2$. The system of equations over D consists of $n + m(n^2+n-1)$ equations in $(2n + m(n-1)(n+2))/2$ variables. Thus, an instance of MQ problem over $\mathbb{F}_2$ is transformed into an instance of MQ problem over D in polynomial-time complexity. The existence of a solution for MQ problem over D implies the existence of a solution for MQ problem over $\mathbb{F}_2$.

The last part of this proof is to show that if addition and multiplication in D can be done in polynomial-time, then the MQ problem over D is NP-complete. Consider the following nondeterministic algorithm to find a solution of an MQ problem over D

1. Select an element $(d_1, \ldots, d_n) \in D^n$ for variables $x_1, \ldots, x_n$.
2. If all m equations over D are satisfied by $(d_1, \ldots, d_n)$ then return `True`. Otherwise, go to step 1.

Note that if addition and multiplication in D can be computed in polynomial-time, then step 2 in the above procedure is also polynomial-time. Therefore, the decisional MQ problem over D, where operations in D are computable in polynomial-time, is NP-complete. Otherwise, the MQ problem over D is NP-hard. ■

2.2 Multivariate Public-Key Cryptography

In Chapter 1 we have discussed the importance of public-key cryptography in solving the problem of key-distribution for symmetric-key primitives using insecure channels. Two of the most successful proposals today are based on the difficulty of integer factorization [RSA78] and computation of discrete logarithms in a cyclic group [Gam84]. Although both problems are regarded as hard problems for classical computers, in 1997 Peter Shor published a polynomial-time quantum algorithm to solve integer factorization and discrete logarithm [Sho97].

This situation has prompted researchers to revisit public-key proposals in the past that were based on different computational hard problems. In 1985, Matsumoto and Imai introduced the first construction of multivariate public-key cryptography in which the security is based on the difficulty of solving an MQ-problem over $\mathbb{F}_2$ [IM85]. Since then, various other public-key cryptosystems based on the hardness of solving systems of polynomial equations over (small) finite field have been proposed [Pat96, PBD14, KPG99, PCDY17, DS05, Din04, TDTD13, CBD+09]. This also has led to several submitted proposals to the NIST post-quantum cryptography standardization process that are based on the hardness of the MQ problem over finite fields: DME [Lue17], RAINBOW [DS05], LUOV [BP17], DUALMODEMS [FPR17a], GEMSS [FPR+17b], GUI [DCP+17], HIMQ-3 [SPK17] and MQDSS [CHR+17]. Despite the devastating attack on RAINBOW [Beu22], which is the multivariate signature scheme that advances the farthest from the initial NIST call-for-proposals for the post-quantum cryptography standardization, the interest in developing other multivariate-based digital signature schemes remains high due to its two primary advantages: small signature size and fast verification. This is reflected in the 10 out of 40 submissions to the NIST call for additional post-quantum digital signature schemes standardization that are classified under the category of multivariate signatures: 3WISE [Rod23a], DME SIGN [Ln23], HPPC [Rod23b], MAYO [BCC+23], PROV [GCF+23], QR-UOV [FIH+23], SNOVA [WCD+23], TUOV [DGG+23], UOV [BCD+23], and VOX [PCF+23]. Moreover, the security of two other submissions that are classified under MPC-in-the-Head signatures, namely BISCUIT [BKPV23] and MQOM [FR23], are also based on the hardness of solving multivariate polynomial equations over finite field.

The public-key of a multivariate-based encryption/signature scheme consists of quadratic polynomials in $F = (f_1, \ldots, f_m) \subset \mathbb{F}_q[x_1, \ldots, x_n]$ (though there is also a construction that uses cubic polynomials [DPW14]). Note that F has three different natures: as an ordered subset of the ring $\mathbb{F}_q[x_1, \ldots, x_n]$, as a system of equations $f_1 = 0, \ldots, f_m = 0$, and as a function $F : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$.

The construction of F as a function is a composition of the following three functions

$$F = B \circ \bar{F} \circ A$$

where $A : \mathbb{F}_q^n \mapsto \mathbb{F}_q^n$ and $B : \mathbb{F}_q^m \mapsto \mathbb{F}_q^m$ are invertible affine transformations and $\bar{F} : \mathbb{F}_q^n \mapsto \mathbb{F}_q^m$ is a quadratic map. The affine functions A and B are randomly selected and their purpose is to hide the algebraic structure of $\bar{F}$ given the knowledge of F. The construction of $\bar{F}$ must satisfy two main requirements. The first one is, given any y in the image of $\bar{F}$, it should be computationally feasible to find $x \in \mathbb{F}_q^n$ such that $\bar{F}(x) = y$. The second requirement is that the complexity of solving $F = B \circ \bar{F} \circ A$ as a system of equations must be as close

as possible to the complexity of solving random quadratic system of equations over $\mathbb{F}_q$. The public-key consists of m polynomial components of F and the private-key consists of A and B. The function $\bar{F}$ may or may not be a part of the private-key and that depends on its exact nature.

Public-key encryption In order to use MQ problems as an encryption function, the function $\bar{F}$ must be injective, i.e., the preimage of every element in the image of $\bar{F}$ must be unambiguously determined. This implies that F is an overdefined system of equations where $m \geq n$. The encryption of a plaintext $P \in \mathbb{F}_q^n$ is performed by computing $C = F(P) \in \mathbb{F}_q^m$. (When $\bar{F}$ is non-injective, one can concatenate the plaintext and its hash value before encryption. The hash value helps to determine the correct preimage when inverting $\bar{F}$.)

The receiver computes $A^{-1}(\bar{F}^{-1}(B^{-1}(C)))$ to recover the original plaintext P. Given F and C, in order to recover the plaintext P without the knowledge of $A, B, \bar{F}$, an attacker must solve the system of equations $F(x_1, \ldots, x_n) = C$.

Public-key signature In the case of an MQ-based digital signature, the function $\bar{F}$ should be close to surjective. This implies that F is an underdefined system of equations where $m \leq n$. A signature S for a plaintext $P \in \mathbb{F}_q^m$ is computed as $S = A^{-1}(\bar{F}^{-1}(B^{-1}(P)))$.

The verifier computes $F(S)$ and checks if $P = F(S)$. In order to forge a signature, an attacker has to solve the polynomial system $F(x_1, \ldots, x_n) = P$.

2.3 Algebraic Cryptanalysis of Symmetric-Key Cryptography

In this section we explain how the problem of solving a system of multivariate polynomial equations over a finite field, especially over $\mathbb{F}_2$, occurs in the cryptanalysis of symmetric-key primitives. We focus particularly on the cryptanalysis of block ciphers and stream ciphers.

2.3.1 Algebraic Cryptanalysis of Block Ciphers

Let $\mathbb{F}_2^n$ be the plaintext/ciphertext space and $\mathbb{F}_2^\kappa$ be the key space. The function $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \mapsto \mathbb{F}_2^n$ is an n-bit block cipher with κ-bit secret key if for every $K \in \mathbb{F}_2^\kappa$ the function

$$\begin{aligned} E_K : \mathbb{F}_2^n &\mapsto \mathbb{F}_2^n \\ P &\mapsto E(P, K) \end{aligned} \tag{2.2}$$

is invertible. A block cipher E is called an *r-round iterated block cipher* if

$$E_K(P) = R_{K_r}(R_{K_{r-1}}(\ldots(R_{K_1}(P)))).$$

where $R : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \mapsto \mathbb{F}_2^n$ is a *round function* of E and R_{K_i} is defined similarly as in (2.2) with respect to R for $i = 1, \ldots, r$ and $K_i \in \mathbb{F}_2^\kappa$. The keys K_i are

called the *round keys* and they are derived from the original key K using a *key-schedule algorithm*. In this subsection, the term block cipher refers to iterated block cipher.

Recall that conventional cryptanalytic approaches to block ciphers are dominated by two techniques: linear cryptanalysis [Mat93] and differential cryptanalysis [BS90]. In the case of linear cryptanalysis an attacker studies linear relations, i.e., the parity of chosen bits of the plaintext, ciphertext, and the key, with high probability. On the other hand, differential cryptanalysis studies the difference between two plaintexts and how this difference evolves in various operations of the block cipher. Many other cryptanalytic techniques for block ciphers are derived from the two techniques mentioned previously [Knu94, KR96, BBS99, LH94, HCN19]. These techniques are all statistical in nature and they require large number of plaintexts and ciphertexts in order to use them to obtain partial or full information about the key with high success probability.

The approach to algebraically mount a key-recovery attack on a block cipher E starts by viewing E as a system of multivariate polynomial equations over $\mathbb{F}_2$. Given a plaintext and its corresponding ciphertext, the indeterminates in this system of equations represent the internal state of the block cipher and the unknown key. Solving this system means recovering full information about the key as well as the internal state. In contrast to other conventional cryptanalysis techniques for block ciphers, this approach is deterministic and it requires only a handful of plaintexts-ciphertexts to obtain full information about the key.

In practice, modelling the encryption function of a block cipher as a system of equations considers each layer of a round function (linear, nonlinear, key addition) separately. For a block cipher with an ℓ layer of operations, the model defines the internal state variables $\mathcal{X}_i = \{x_{i,1}, \ldots, x_{i,n}\}, i = 0, 1, \ldots, \ell$. The set $\mathcal{X}_i$ has dual roles: it represents the output variables of the i-th layer and the input variables for the $(i+1)$-th layer. The set $\mathcal{X}_0$ and $\mathcal{X}_\ell$ are the set of plaintext and ciphertext variables, respectively. Let $\mathcal{K} = \{k_1, \ldots, k_\kappa\}$ denote the set of key variables. If the round keys are generated by selecting a subset of bits from the master key K, then $\mathcal{K}$ is the set of all key variables in the system of equations. Otherwise, for each $i = 1, \ldots, r$ and for some $j \in \{1, \ldots, n\}$, the model defines the set $\mathcal{K}_i = \{k_{i,1}, \ldots, k_{i,j}\}$ as the variables for the i-th round key. The system of equations for a block cipher can then be divided into three disjoint sets: (1) a set of linear equations that represent the linear layers and key addition, (2) a set of non-linear equations describing the non-linear operations of the cipher, and (3) a set of equations that describes the key-schedule algorithm.

Generating a system of equations for linear layers is rather obvious. For the non-linear layers, block ciphers in general employ a nonlinear map $S : \mathbb{F}_2^s \mapsto \mathbb{F}_2^t$ with relatively small s, t commonly referred to as *substitution boxes* (S-Box). We describe two approaches to obtain a system of polynomial equations that represent S. We denote by x_i, y_j $(1 \leq i \leq s, 1 \leq j \leq t)$ the input and output variable of S respectively. An S-Box S can be viewed as a parallel applications of t s-variable Boolean functions i.e, $S(x) = (h_1(x), \ldots, h_t(x))$ where $h_i : \mathbb{F}_2^s \mapsto \mathbb{F}_2$ for $i = 1, \ldots, t$. The first approach to obtain a set of polynomials that represent S is to compute the *algebraic normal form* of h_i for each $i = 1, \ldots, t$ (see [Car10, pg. 9] for the definition and the algorithm to compute the algebraic normal form of a Boolean function). An example of this approach is given in Figure 5. The second approach is described in [BC03] and the goal is to obtain as many linearly

independent polynomial equations as possible. Firstly, it selects $d \in \mathbb{Z}$ such that $\sum_{i=0}^{d} \binom{s+t}{i} > 2^s$. In order to find all linearly independent equations involving monomials of degree $\leq d$, we construct a $\sum_{i=0}^{d} \binom{s+t}{i} \times (2^s+1)$ matrix containing a separate row for each monomial, where the last column of the matrix consists of these monomials. The entries from column 1 to the column 2^s corresponding to the different values of the particular monomial for all possible input and output values. The final step is to compute the (reduced)-row echelon form of the matrix and all row operations required by this step are applied to the corresponding monomials as well. In this way, the zero rows appear in the (reduced)-row echelon form of the matrix and all the polynomials corresponding to these zero rows are a set of linearly independent equations representing the S-Box. We provide an example for both approaches in Figure 6.

$$
\begin{aligned}
&y_1 + x_1x_2 + x_1x_3 + x_1 + x_2x_3 + x_2 + 1,\\
&y_2 + x_1x_3 + x_2 + 1,\\
&y_3 + x_1 + x_2x_3 + x_2 + x_3 + 1
\end{aligned}
$$

Figure 5: *An example of a system of equations representing the S-Box defined by the lookup table* $(7, 6, 0, 4, 2, 5, 1, 3)$ *by computing the algebraic normal form of each coordinate function.*

$$
\left[\begin{array}{cccccccc|l}
1&1&1&1&1&1&1&1&1\\
0&1&0&1&0&1&0&1&x_1\\
0&0&1&1&0&0&1&1&x_2\\
0&0&0&0&1&1&1&1&x_3\\
1&0&0&0&0&1&1&1&y_1\\
1&1&0&0&1&0&0&1&y_2\\
1&1&0&1&0&1&0&0&y_3\\
0&0&0&1&0&0&0&1&x_1x_2\\
0&0&0&0&0&1&0&1&x_1x_3\\
0&0&0&0&0&1&0&1&x_1y_1\\
0&1&0&0&0&0&0&1&x_1y_2\\
0&1&0&1&0&1&0&0&x_1y_3\\
0&0&0&0&0&0&1&1&x_2x_3\\
0&0&0&0&0&0&1&1&x_2y_1\\
0&0&0&0&0&0&0&1&x_2y_2\\
0&0&0&1&0&0&0&0&x_2y_3\\
0&0&0&0&0&1&1&1&x_3y_1\\
0&0&0&0&1&0&0&1&x_3y_2\\
0&0&0&0&0&1&0&0&x_3y_3\\
1&0&0&0&0&0&0&1&y_1y_2\\
1&0&0&0&0&1&0&0&y_1y_3\\
1&1&0&0&0&0&0&0&y_2y_3
\end{array}\right]
\rightarrow
\left[\begin{array}{cccccccc|l}
1&0&0&0&0&0&0&0&x_1y_3+y_3\\
0&1&0&0&0&0&0&0&x_1y_3+x_1+x_3+y_1+y_2\\
0&0&1&0&0&0&0&0&x_1y_3+x_1+x_2+x_3+y_2+y_3\\
0&0&0&1&0&0&0&0&x_1y_3+x_2+x_3+y_1+1\\
0&0&0&0&1&0&0&0&x_1y_3+x_3+y_1+y_3\\
0&0&0&0&0&1&0&0&x_1y_3+x_1+x_2+y_2+1\\
0&0&0&0&0&0&1&0&x_1y_3+x_2+y_1+y_2+y_3+1\\
0&0&0&0&0&0&0&1&x_1y_3+x_1\\
\hline
0&0&0&0&0&0&0&0&x_1x_3+x_2+y_2+1\\
0&0&0&0&0&0&0&0&x_1y_1+x_2+y_2+1\\
0&0&0&0&0&0&0&0&x_1y_2+x_3+y_1+y_2\\
0&0&0&0&0&0&0&0&x_1x_2+x_1+x_2+x_3+y_1+1\\
0&0&0&0&0&0&0&0&x_2x_3+x_1+x_2+y_1+y_2+y_3+1\\
0&0&0&0&0&0&0&0&x_2y_1+x_1+x_2+y_1+y_2+y_3+1\\
0&0&0&0&0&0&0&0&x_1y_3+x_2y_2+x_1\\
0&0&0&0&0&0&0&0&x_1y_3+x_2y_3+x_2+x_3+y_1+1\\
0&0&0&0&0&0&0&0&x_1y_3+x_3y_1+y_1+y_3\\
0&0&0&0&0&0&0&0&x_3y_2+x_1+x_3+y_1+y_3\\
0&0&0&0&0&0&0&0&x_1y_3+x_3y_3+x_1+x_2+y_2+1\\
0&0&0&0&0&0&0&0&y_1y_2+x_1+y_3\\
0&0&0&0&0&0&0&0&y_1y_3+x_1+x_2+y_2+y_3+1\\
0&0&0&0&0&0&0&0&y_2y_3+x_1+x_3+y_1+y_2+y_3
\end{array}\right]
$$

Figure 6: *An example of a system of equations representing the S-Box defined by the lookup table* $(7, 6, 0, 4, 2, 5, 1, 3)$ *using the method described in [BC03]. The zero-rows correspond to the desired polynomial equations for the S-Box.*

In order to restrict the set of solutions to the base field $\mathbb{F}_2$, we add polynomials of the form $x^2 + x$ for all variables x in the system of equations. This is equivalent to working in the quotient ring $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle$. Magma [BCP97] and PolyBoRi/BRiAl [BD09b] offer dedicated implementation for polynomial computations in $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle$.

The systems of equations arising from block ciphers usually have large num-

ber of variables and equations that are sparse and highly structured. Table 3 describes the size of the equation system from several block ciphers such as MISTY1 [Mat97], KHAZAD [BR00], KASUMI [Pro99], and CAMELLIA-128 [AIK+00] and SERPENT-128 [BAK98].

	# variables	# equations
KHAZAD	6464	7664
MISTY1	3856	3856
KASUMI	4264	4264
CAMELLIA-128	3584	6224
SERPENT-128	16640	17680

Table 3: *The size of equation system from several block ciphers [BC03].*

Note that due to the large number of variables and equations, the direct approach to solve a system of polynomial equations from a block cipher often turns out to be infeasible. A more promising approach is to combine generic methods to solve polynomial equations together with techniques that exploit the structural properties of block ciphers.

One direction is to employ a *divide-and-conquer* approach. For instance, the iterative nature of a block cipher allows the equations to be splitted into several subsystems. Let r be the number of rounds and assume that r is even. We split the equations into two subsystems represent the first and the last $r/2$ rounds. The input variables of the second subsystem correspond to the output variables of the first subsystem. The goal is to eliminate variables that do not appear in the round $r/2$ and $r/2+1$ by computing the Gröbner bases of the first and the second subsystem independently (see Chapter 3 for the definition of Gröbner bases). This *meet-in-the-middle* approach was proposed and tested by Cid et al. against small-scale variants of Advanced Encryption Standard (AES) [CMR05, CMR06]. Another divide-and-conquer approach, proposed by Albrecht [Alb07], is based on an incremental method. The idea is to compute the Gröbner basis of the whole system round-by-round, starting from the first round. A Gröbner basis corresponding to the ideal generated by polynomials at the i-th round is added to the set of polynomials for the $(i+1)$-th round. This is iteratively applied until the system of equations for the last round.

Another approach in the algebraic cryptanalysis on block ciphers is to combine it with conventional cryptanalysis techniques such as linear or differential cryptanalysis. Albrecht and Cid [AC09] proposed an attack that combines an algebraic attack with differential cryptanalysis. If an r'-rounds differential characteristic was found, to perform a key-recovery attack typically an attacker tries to guess several key bits in the last $r_d = r - r'$ rounds for an r-rounds block cipher. The authors used algebraic techniques to extend r_d from 2 to 4 rounds for the block cipher PRESENT [BKL+07]. The proposed method is to construct polynomial equations for many plaintext pairs and connect pairs using a set of linear equations that represent a difference in the plaintext. A Gröbner basis algorithm is used to check if the equations satisfy the differential characteristic, instead of directly performing a key-recovery attack.

2.3.2 Algebraic Cryptanalysis of Stream Ciphers

Even though algebraic cryptanalysis did not receive much success in attacking block ciphers, it is considered to be an effective approach to attack several classes of stream ciphers [CM03, Cou03]. The goal of algebraic cryptanalysis of stream ciphers is to recover its initial state from some subset of the keystream. The attack model assumes that the attacker has access to some keystream bits and they need not even be consecutive.

There are two classical stream cipher constructions based on LFSR. The first construction uses multiple LFSRs in parallel and combines their outputs with a Boolean function. We refer to this construction as a *combination generator*. Another construction employs one LFSR together with a Boolean function with the LFSR state as its input to generate an output sequence. This construction is known as a *filter generator*. We refer to the Boolean function used to filter the output of multiple LFSRs in a combination generator or the internal state of the stream cipher in a filter generator as a *non-linear filter*. A standard cryptographic criterion for a non-linear filter is that its algebraic degree should be sufficiently high. The high degree of the non-linear filter provides resistance against algebraic cryptanalysis. In [CM03], Courtois and Meier introduced a simple yet a powerful strategy to significantly reduce the degree of the polynomials from a combination generator or a filter generator. The main idea is to multiply each polynomial by a well-chosen nonzero polynomial such that the resulting product is of low degree.

In this section, we outline the main idea behind the attack of Courtois and Meier. We restrict ourselves to stream ciphers in which the state and the keystream are elements of $\mathbb{F}_2$ and they generate one output bit at a time. The Boolean function that computes the output bit from the internal state of the stream cipher is denoted by f. The function L that computes the next state of a stream cipher is a linear transformation over $\mathbb{F}_2$ and we assume L is public. Note that the computation on polynomials representing stream ciphers is done on the ring $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$.

Let $s_1, \ldots, s_n$ be variables that represent the initial state of a stream cipher and let z_t denote the variables that represent the output bit generated at the time t. The polynomial equation of the stream cipher that represent the computation of the output bit at time t is given by

$$f(L^t(s_1, \ldots, s_n)) = z_t \qquad t = 0, 1, 2, \ldots.$$

The main idea of Courtois-Meier attack is to find a well-chosen multivariate polynomial g such that multiplying g by $f(L^t(s_1, \ldots, s_n)) + z_t$ yields a substantially lower degree polynomial. For instance, when $z_t = 0$ then we can get an equation $f(L^t(s_1, \ldots, s_n))g(L^t(s_1, \ldots, s_n)) = 0$ of degree lower than the degree of $f(L^t(s_1, \ldots, s_n))$. With sufficiently many keystream bits, an attacker can obtain an overdefined system of equations that is efficiently solvable. Note that in case $z_t = 1$, then g is also required to be a low degree polynomial. We may also use different g for the same f in order to generate more polynomial equations. The polynomial g is called the *annihilator* of f. The minimum degree of g such that g is an annihilator of f is called the *algebraic immunity* of f. Courtois and Meier proved that there exists such g of degree less than or equal to $\lceil n/2 \rceil$ such that the degree of fg is at most $\lceil (n+1)/2 \rceil$ (see Theorem 6.0.1 in [CM03]). We refer to the Section 9 of [Car10] for further discussion about algebraic im-

munity, its relation to other cryptographic criteria of Boolean functions, and several known functions that have high algebraic immunity.

The next question arises is how to find such g. One strategy is to first consider terms of high degree in f and see if they share a common non-trivial low degree divisor g'. The product fg with $g = g' - 1$ yields cancellation of terms of f divisible by g', which shows that the polynomial fg is of low degree.

Courtois and Meier demonstrated their idea to attack the stream cipher Toyocrypt(see [DCG+01, Section 2] for a full description of ToyoCrypt). It is a filter generator stream cipher with one 128-bit LFSR and a non-linear Boolean function of the form

$$f(x_1, \ldots, x_{128}) = x_{128} + \sum_{i=1}^{63} x_i x_{\alpha_i} + x_{11}x_{24}x_{33}x_{43} + x_2x_3x_{10}x_{13}x_{19}x_{21}x_{24}x_{26}x_{27}x_{29}x_{34}x_{39}x_{42}x_{43}x_{52}x_{54}x_{60} + \prod_{i=1}^{63} x_i$$

with $\{\alpha_1, \ldots, \alpha_{63}\}$ being some permutation of the set $\{64, \ldots, 126\}$. Observe that that the terms of degree 4, 17, and 63 have a common factor $x_{24}x_{43}$. Let $g = x_{24} - 1$. Multiplying f by g yields a cubic polynomial since all the terms of f divisible by x_{24} are cancelled out. Similarly, the polynomial $f \cdot (x_{43} - 1)$ is also of degree 3 by the same reasoning. Thus, for each keystream bit, we can generate two cubic polynomials in variables $s_1, \ldots, s_{128}$.

Let m be the number of known keystream bits, which corresponds to the number of polynomials. Once an attacker obtains $m \geq \sum_{i=0}^{d} \binom{n}{i}$ keystream bits, the initial state of the stream cipher is recoverable in a polynomial-time using standard techniques from linear algebra where each monomial in the system of equations is treated as an independent variable.

In case of Toyocrypt, there are $T = \sum_{i=0}^{3} \binom{128}{i}$ monomials of degree less than or equal to 3. Recall that for each keystream bit we obtain two linearly independent cubic polynomials. By obtaining $T/2$ keystream bits, the initial state can be recovered in T^ω bit operations where $2 \leq \omega \leq 3$ is the linear algebra constant.*

A similar attack is also applied against Lili-128, an irregularly clocked stream cipher consisting two LFSRs of size 39-bit and 89-bit [SDGM00]. The first LFSR of size 39-bit is used to clock the second LFSR, in which the output is filtered by a non-linear filter f. The non-linear filter in Lili-128 is a 10-variable Boolean function of degree 6. The algebraic normal form of the non-linear filter is the following

$$\begin{aligned} f(x_1, \ldots, x_{10}) = {} & x_1x_8x_9x_{10} + x_1x_8 + x_1x_9 + x_2x_7x_8x_9 + x_2x_7x_9x_{10} + x_2x_8 + \\ & x_2x_9x_{10} + x_2 + x_3x_7x_8x_9x_{10} + x_3x_7x_8x_{10} + x_3x_7x_9 + x_3x_8x_9x_{10} + x_3x_8x_9 + \\ & x_3x_8x_{10} + x_3x_9x_{10} + x_3x_9 + x_3 + x_4x_6x_7x_8x_9x_{10} + x_4x_6x_7x_8x_9 + \\ & x_4x_6x_7x_9x_{10} + x_4x_6x_7x_9 + x_4x_7x_8x_9x_{10} + x_4x_7x_8x_9 + x_4x_7x_9x_{10} + x_4x_7x_9 + \\ & x_4x_7x_{10} + x_4x_8x_9x_{10} + x_4x_8x_{10} + x_4x_9x_{10} + x_4x_{10} + x_4 + x_5x_6x_7x_8x_9x_{10} + \\ & x_5x_6x_7x_8x_9 + x_5x_6x_7x_9x_{10} + x_5x_6x_7x_9 + x_5x_7x_8x_{10} + x_5x_7x_{10} + x_5x_9x_{10} + \\ & x_5 + x_6x_7x_9x_{10} + x_6x_7x_9 + x_6x_7x_{10} + x_6x_7 + x_6x_8x_9x_{10} + x_6x_8x_9 + x_6x_{10}. \end{aligned}$$

* For instance, $\omega = 3$ when uses Gaussian elimination.

The terms of degree 5 and 6 of f have a common divisor x_7x_9. Thus, the polynomials $f \cdot (x_7 - 1)$ and $f \cdot (x_9 - 1)$ have degree 5. Moreover, the terms of degree 4 and 5 in $f \cdot (x_9 - 1)$ have a common factor x_{10}. Thus, the degree of the polynomial $f \cdot (x_9 - 1) \cdot (x_{10} - 1)$ is equal to 4. In [CM03] Courtois and Meier found 14 linearly independent polynomials g of degree 4 such that the polynomial fg also has degree 4.

The attack for Lili-128 proceeds as follows. Since the first 39-bit LFSR is used to clock the second LFSR, we need to guess the internal state of the first LFSR which has 2^{39} possible states. The degree of the polynomials are reduced from degree 6 to degree 4, hence there are $T = \sum_{i=0}^{4} \binom{89}{i}$ possible monomials of degree ≤ 4. For each keystream bit, we obtain 14 linearly independent equations. It is then sufficient to obtain $T/14$ keystream bits to recover the initial state. The time complexity of the attack is then equal to $2^{39} \cdot T^{\omega}$ where ω is the linear algebra constant.

The same attack can also be applied against E0, a stream cipher which is part of the Bluetooth scheme used for wireless communications [AK03, Blu01]. The stream cipher E0 is a non-linear combiner with a 4-bit memory and 4 LFSRs with a total length of 128-bits. Armknecht and Krause showed that given 4 consecutive keystream bits, there exists a polynomial equation of degree 4 [AK03]. The number of polynomials of degree ≤ 4 is equal to $T = \sum_{i=0}^{4} \binom{128}{i}$. In order to minimize the amount of data needed, the attacker must have access to $T + 3$ consecutive keystream bits. Otherwise, the algebraic cryptanalysis on E0 requires access to T 4-bit consecutive keystream bits, which in total requires $4T$ keystream bits. Note that this is different from the previous attack on Toyocrypt and Lili-128 that does not require access to consecutive keystream bits.

In [Cou03] Courtois described a new approach to reduce the complexity of algebraic attacks on stream ciphers called *fast algebraic attack*. The attack model assumes that the attacker is able to obtain consecutive keystream bits. Once obtained, the system of equations involved will have a recursive structure. With such structure, Courtois proposed solving the system of equations using Berlekamp-Massey algorithm, which has a linear complexity instead of Gaussian elimination that has cubic complexity.

2.4 Algebraic Cryptanalysis of Post-Quantum Cryptography

When applied to post-quantum cryptographic (PQC) schemes, i.e., schemes that are designed to withstand quantum adversaries, algebraic cryptanalysis also serves as a valuable analytical tool. PQC schemes includes not only MPKCs, but also code-based, lattice-based, hash-based, and isogeny-based constructions. For an overview of code-based, lattice-based and hash-based schemes, we refer the reader to [BBD09], while the overview on isogeny-based schemes can be found in [JDF11, Feo17].

The primary targets of algebraic cryptanalysis within PQC schemes are naturally the MPKCs. Nonetheless, its applicability extends beyond MPKCs, as it has also proven relevant in the context of code-based and lattice-based schemes. However, its impact tends to be more limited for isogeny-based and hash-based schemes.

Based on the attack objective, the algebraic cryptanalysis against MPKCs can be categorized into two types. The first type, direct attacks, aims at recov-

ering plaintexts for encryption schemes or forging signatures for digital signature schemes. These attacks directly attempt to find a solution to the system of polynomial equations from the public-key of the schemes [FJ03]. The second type, key-recovery attacks, aim to recover the private key of the schemes. This involves solving instances of the MinRank problem (see [BBD09, pg. 224] for a detailed description). In many approaches, a critical step towards solving MinRank instances requires solving systems of polynomial equations, as demonstrated in [FGP$^+$15, AST24, Beu22, Beu24].

In code-based schemes, algebraic cryptanalysis generally models the underlying decoding problem, or closely related structural problems, as systems of polynomial equations. For instance, the algebraic attack in [BBB$^+$20] presents improvements over previous security assumptions. Similarly, the work in [FOPT10] demonstrates that certain attempts to reduce the key size lead to a practical algebraic attack that recovers the private key. In contrast, lattice-based PQC schemes exhibit strong resistance against algebraic attacks. Nonetheless, several works have explored the use of algebraic cryptanalysis to solve the computational problem underpinning lattice-based PQC schemes [AG11, ACF$^+$15, Ste24].

2.5 Solving Systems of Polynomials in Algebraic Cryptanalysis

2.5.1 Gröbner Bases

One of the most widely used methods to solve systems of polynomial equations is using a Gröbner basis algorithm. The theory of Gröbner bases treats a system of polynomial equations as generators for an ideal in the polynomial ring, instead of merely as a set of polynomials. The rationale behind this is that a solution for the system of equations is also a solution for any polynomial in the ideal. Additionally using an appropriate ordering on the monomials, the set of solutions for the system of equations can be parametrized by some polynomials in the ideal.

Gröbner basis algorithms compute a so-called Gröbner basis of the ideal generated by the polynomials in the equation system, with respect to a chosen ordering on the set of monomials. The theory of Gröbner bases was developed by Bruno Buchberger in his PhD thesis [Buc65, Buc06]. One of its immediate application is that Gröbner bases allows us to determine whether a polynomial in the ring is also an element of the ideal. In this regard, Gröbner bases is seen as the multivariate generalization of the greatest common divisor of univariate polynomials.

On the other hand, computing a Gröbner basis for the ideal equipped with the appropriate monomial ordering allows us to recover a set of solutions for the system of polynomial equations that generates the ideal. In this respect, Gröbner bases algorithm is seen as a nonlinear generalization of Gaussian elimination on a set of linear equations. The theory of Gröbner bases together with its applications are discussed further in Chapter 3.

Given an arbitrary basis of a polynomial ideal, there exist algorithms that compute a Gröbner basis of the ideal. Buchberger proposed the first algorithm to compute a Gröbner basis of an ideal in his PhD thesis [Buc65, Buc06]. A major improvement in this field was the F_4 algorithm proposed by Jean-Charles

Faugère [Fau99]. Faugère devised a method to perform reduction on multiple polynomials at once using fast linear algebra. In Chapter 4 we will discuss both algorithms in more detail.

2.5.2 Linearization Based Algorithms

Another technique to solve a system of quadratic polynomial equations that received a lot of attention from cryptography community is linearization-based algorithms. The main idea of linearization-based algorithms is to first interpret each monomial occurring in the system of equations as a new variable. Once linearized and solved, the solutions of the linearized system of equations is then validated against the original system of polynomial equations.

Since the problem of solving an instance of MQ problem is now reduced to the problem of solving a system of linear equations, the efficiency of linearization-based algorithms relies on the number of linearly independent polynomials in the system. For example, in an n-variables polynomial ring over $\mathbb{F}_2$ there are $\binom{n}{2} + n$ monomials of degree less than or equal to 2 (considering $x^2 = x$ and ignoring constant monomial). Thus, if an instance of MQ problem over $\mathbb{F}_2$ in n variables has $\binom{n}{2} + n$ linearly independent equations, then the linearization method works best for this particular case. There are many algorithms that are based on linearization technique [CKPS00, CP02, CP03, Cou04, MP08, CB07, MDBW09, MMDB08, MCD$^+$09]. Most of the variants focus on generating sufficiently many linearly independent equations.

One of the prominent example is *eXtended Linearization* (XL) algorithm, which was introduced in [CKPS00]. XL algorithm takes a system of polynomial equations and a positive integer D as inputs. It generates more polynomials by multiplying a polynomial f in the initial system of equations by all monomials $\mathfrak{m}$ such that the degree of $\mathfrak{m}f$ is less than or equal to D. Once all polynomials of the form $\mathfrak{m}f$ are added to the initial system of equations, the algorithm proceeds by linearizing the system and tries to find solutions to the linearized system of equations. Later XL turned out to be closely related to Gröbner bases algorithms, in particular the F_4 algorithm as shown in [AFI$^+$04]. A more detailed discussion on the XL algorithm and its F_4-like description are given in Section 4.3.

The simplicity of XL eventually stimulates the development of other variants. One of them is *eXtended Sparse Linearization* (XSL) [CP02]. Instead of multiplying a polynomial f in the initial system by all possible monomials of degree less than or equal to $D - 2$, XSL algorithm multiplies f by "carefully selected monomials" [CP02]. The aim is to avoid introducing unnecessary monomials when generating new equations by taking advantage of the sparsity and the structure of the system of equations. However, the authors did not explicitly state how the monomials should be chosen, leaving some rooms for interpretation.

In [MP08], Murphy and Paterson provide a generalization of the XL algorithm. In the paper the authors proved that the XL algorithm is a specialization of an algorithm that finds the intersection of hyperplanes called *GeometricXL*. The main idea of GeometricXL is based on the fact that the formulation of a MQ problem and its solution are invariant under a linear coordinate transformation. This is particularly relevant especially in the case of multivariate public-key cryptosystems where linear changes of coordinates are required in

order to hide the structure of the central map. However, the algorithm requires the order of the underlying finite field to be larger than the maximal degree D.

Another variant of XL called *MutantXL* was introduced in [MMDB08, MCD+09, MDBW09]. The authors proposed a concept of *mutants*. They are the polynomials that have degree strictly less than D after the elimination step. However, it turns out that the concept of mutant is closely related with *normal selection strategy* used in the F_4 algorithm. Thus the MutantXL algorithm can be fundamentally understood as a redundant variant of the F_4 algorithm [ACFP12].

Another linearization based approach is the Crossbred algorithm, proposed by Joux and Vitse [JV17]. It is a hybrid method for solving systems of multivariate polynomial equations over finite fields, particularly $\mathbb{F}_2$, that combines variable partitioning and partial enumeration with XL-like linearization. Given positive integers D, d, k, the algorithm first generates r new polynomials of degree D and degree d in the first k variables, which are then appended to the original system. Then it performs an exhaustive search over all possible values $\mathbb{F}_2^{n-k}$ for the last $n-k$ variables. For each assignment to the last $n-k$ variables, the resulting system in the first k variables is solved. If it has no valid solution then the exhaustive search continues until a valid solution is found [VDI24].

2.5.3 SAT Solving

Recall that the proof of NP-completeness of MQ over $\mathbb{F}_2$ in Proposition 2.4 is based on the reduction of a 3-CNF SAT instance into an instance of multivariate quadratic polynomials over $\mathbb{F}_2$. One may then ask the following question: can we solve an instance of the MQ problem over $\mathbb{F}_2$ by reducing it into an instance of the SAT problem in Conjunctive Normal Form ?

In this subsection we discuss how a problem of solving an instance of MQ problem over $\mathbb{F}_2$ is viewed as a SAT problem. We mention two existing strategies in the literature to convert a system of polynomial equations over $\mathbb{F}_2$ into a SAT problem in CNF.

The first strategy is based on linearization of polynomials and conversion of the resulting linear polynomial into CNF. Since this strategy is proven to be effective for dense polynomial, we refer to this as *dense conversion*. The second conversion strategy is based on the truth table of the polynomials. The method relies on the number of variables in the polynomial being relatively small, i.e., the polynomials must be sparse. Thus we refer to the second strategy as *sparse conversion*. We will now discuss them both in depth.

Dense conversion This idea of converting an instance of MQ over $\mathbb{F}_2$ into a SAT problem in CNF was first proposed by Courtois, Bard, and Jefferson in [BCJ07]. It is also mentioned in Gregory Bard's PhD thesis [Bar07] and in Chapter 13 of [Bar09]. The dense conversion strategy is based on the observation that the derivation of the CNF formula for a linear polynomial over $\mathbb{F}_2$ is straightforward. For instance, the polynomial $x_1 + x_2 + x_3 + x_4$ is represented by the following Boolean formula in CNF

$$\begin{aligned}
&(X_1 \vee X_2 \vee X_3 \vee X_4) \wedge (X_1 \vee X_2 \vee \neg X_3 \vee \neg X_4) \wedge \\
&(X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4) \wedge (X_1 \vee \neg X_2 \vee \neg X_3 \vee X_4) \wedge \\
&(\neg X_1 \vee X_2 \vee X_3 \vee \neg X_4) \wedge (\neg X_1 \vee X_2 \vee \neg X_3 \vee X_4) \wedge \\
&(\neg X_1 \vee \neg X_2 \vee X_3 \vee X_4) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4).
\end{aligned} \tag{2.3}$$

The dense conversion converts a quadratic polynomial over $\mathbb{F}_2$ into a Boolean formula in CNF in three steps.

The first step is to linearize the quadratic polynomial by replacing the quadratic monomials of the form $x_i x_j$ using additional variables $x_{i,j}$.* An equation of the form $x_i x_j + x_{i,j}$ is then added to the system. Once a linear polynomial is constructed, one may be tempted to directly convert it into CNF. This yields a Boolean formula where the number of literals in each clause is equal to the number of terms, say ℓ, in the linear polynomial. At the same time, the number of clauses generated is equal to $2^{\ell-1}$. Thus, this is not a desirable approach since the large number of clauses and literals in each clause increases the complexity of the SAT solving algorithm.

The second step in dense conversion reduces the number of literals in a clause by cutting each sum in the linear equation into subsums of length, say c. We refer to c as the cutting number. The cutting number represents the number of terms in each subsum, which corresponds to the number of literal in a clause once these subsums are converted to CNF. For instance, let $c = 4$ and $\ell \equiv 2 \pmod{c}$. The linear equation $x_1 + \cdots + x_\ell = 0$ is converted into the following system of linear equations

$$\begin{aligned} x_1 + x_2 + x_3 + y_1 &= 0 \\ y_1 + x_4 + x_5 + y_2 &= 0 \\ &\vdots \\ y_i + x_{4i+2} + x_{4i+3} + y_{i+1} &= 0 \\ &\vdots \\ y_{\lceil \ell/c \rceil - 2} + x_{\ell-2} + x_{\ell-1} + x_\ell &= 0. \end{aligned}$$

where $y_1, \ldots, y_{\lceil \ell/c \rceil - 2}$ are additional auxiliary variables. If $\ell \not\equiv 2 \pmod{c}$ then the number of terms in the last equation is less than c. This implies that the number of literals in the clause that corresponds to the last linear equation is also less than c.

The third step, which is the last step, is to convert each cutted linear equation into a Boolean formula in CNF, similar to Equation 2.3. For a linear equations with ℓ number of terms, there are $\lceil \ell/c \rceil - 1$ subsums and each of it corresponds to a set of 2^{c-1} clauses with c literals. For quadratic polynomials of the form $x_i x_j + x_{i,j}$ that represent the substitution of quadratic monomimals $x_i x_j$ by a dummy variable $x_{i,j}$, we can examine its truth table to derive its corresponding Boolean formula in CNF. For instance, the CNF formula for $x_i x_j + x_{i,j}$ is equal to

$$(X_i \vee \neg X_{i,j}) \wedge (X_j \vee \neg X_{i,j}) \wedge (\neg X_i \vee \neg X_j \vee X_{i,j})$$

Sparse conversion The second approach to convert an instance of the MQ problem over $\mathbb{F}_2$ to a Boolean formula in CNF is due to Brickenstein and

* Note that since Boolean formulas in CNF do not have any constant, an additional Boolean variable, say T, must also be introduced to represent the constant term of the polynomial together with a clause (T) (or $(T \vee T \vee \cdots T)$). In this description, we assume that the quadratic polynomial does not have a constant term.

it is described in his PhD thesis [Bri10]. The conversion is essentially truth-table based, thus requiring the number of variables in the polynomial to be small enough such that it is feasible to obtain all solutions of the polynomial. We demonstrate the sparse conversion strategy in the following example.

Example. *Let* $f = x_1x_3 + x_1 + x_2 + x_3 + 1 \in \mathbb{F}_2[x_1, x_2, x_3]$. *We define the following set*

$$S_f = \{(v_1, v_2, v_3) \in \mathbb{F}_2^3 : f(v_1, v_2, v_3) = 1\}.$$

For each vector $(v_1, v_2, v_3) \in S_f$ *we construct a clause that eliminates the corresponding assignment to be a solution for the Boolean formula in CNF. In this example we have*

$$S_f = \{(0, 0, 0), (1, 1, 0), (0, 1, 1), (1, 1, 1)\}.$$

Therefore, the corresponding Boolean formula for f *in CNF is*

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3) \wedge (X_1 \vee \neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3).$$

Note that the CNF representation of f *above with sparse conversion is not unique. For instance, we can further simplify the CNF formula of* f *as follows*

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2).$$

The two approaches above can be seen as a companion to one another. Therefore, a better algorithm to convert an instance of the MQ problem over $\mathbb{F}_2$ into a SAT problem in CNF combines both techniques. If it encountered a sparse polynomial in the system, the sparse conversion technique is employed to obtain a more compact CNF. Otherwise the polynomial is considered dense and the algorithm then uses dense conversion.

Once constructed, one can feed the CNF into an off-the-shelf SAT solver to obtain a solution. Recall that the SAT problem is a well-known NP-complete problem. This means that the worst-case time complexity is expected to be exponential. Nevertheless, there are many good implementations of SAT solver available such as MINISAT [ES03], CRYPTOMINISAT [SNC09], PLINGELING [Bie17]. In particular, CRYPTOMINISAT is designed to efficiently work on cryptographic related problems. More importantly, it provides an extension to the input language by supporting XOR clauses.

Note that the algorithms to solve instances of the SAT problem in CNF are randomized, implying that a careful approach has to be taken to estimate their average case runtime to solve a particular instance.

2.5.4 Raddum-Semaev Algorithm

In [RS06], Raddum and Semaev introduced another approach to solve a system of equations over $\mathbb{F}_2$. The approach works for sparse systems of equations such that the number of variables that appear in each polynomial is relatively small. We refer to this approach as the Raddum-Semaev algorithm.

Let $f_1, \ldots, f_m$ be polynomials with coefficients in $\mathbb{F}_2$ in variables $X = (x_1, \ldots, x_n)$. Let $\mathsf{Var}(f_i) \subseteq X$ be an ordered set of the variables that appear in

f_i. Assume that for all $i = 1, \ldots, m$, we have $|\mathsf{Var}(f_i)| \leq k$ for some positive integer k such that 2^k is relatively small. Raddum-Semaev algorithm does not treat each polynomial as an element of the ring $\mathbb{F}_2[x_1, \ldots, x_n]$, but simply as an equation restricted to variables that appear in each polynomial.

Definition 2.6. *A configuration of a polynomial f_i is an assignment of values to the variables in $\mathsf{Var}(f_i)$ such that $f_i = 0$.*

Note that the elements in $\mathsf{Var}(f_i)$ are ordered and a configuration of f_i is represented by a bit string of length $|\mathsf{Var}(f_i)|$. The j-th bit of a configuration then corresponds to the value for the j-th variable in $\mathsf{Var}(f_i)$. This allows each polynomial f_i to be identified with $\mathsf{Var}(f_i)$ and a set of configurations of f_i.

Definition 2.7. *A symbol $S_i = (\mathsf{Var}(f_i), L_i)$ of the polynomial f_i consists of $\mathsf{Var}(f_i)$ and a set L_i of configurations of f_i.*

Since the cardinality of $\mathsf{Var}(f_i)$ is small, it is possible to construct L_i by exhaustively running through all bit strings of length $|\mathsf{Var}(f_i)|$ and checking whether it satisfies as a configuration of f_i.

Every solution $s = (s_1, \ldots, s_n) \in \mathbb{F}_2^n$ of the system of equations $f_1, \ldots, f_m$ also satisfies each polynomial f_i for all $i = 1, \ldots, m$. This implies that s restricted to the variables in $\mathsf{Var}(f_i)$ is a configuration of f_i, i.e., it is an element of L_i.

The idea of the Raddum-Semaev algorithm is to repeatedly remove configurations that contradict a solution for the system of equations. Once all pathological configurations are removed, the values of the remaining configurations constitute as candidate solutions of the system.

Definition 2.8. *Let $\mathsf{Var}(f_i) \subseteq \mathsf{Var}(f_j)$ for $i, j \in \{1, \ldots, m\}$. A configuration of f_j* covers *a configuration of f_i if the two are equal for all variables in $\mathsf{Var}(f_i)$.*

Definition 2.9. *Let S_i, S_j be the two symbols of the polynomials f_i, f_j, respectively, where $f_i \neq f_j$. Let $f_{i,j}$ be a polynomial with variables*

$$\mathsf{Var}(f_{i,j}) = \mathsf{Var}(f_i) \cap \mathsf{Var}(f_j)$$

and the symbol $S_{i,j}$ of $f_{i,j}$ is defined as

$$S_{i,j} = \left(\mathsf{Var}(f_{i,j}), \mathbb{F}_2^{|\mathsf{Var}(f_{i,j})|}\right).$$

Let $L_{i,j}, L_{j,i} \subseteq \mathbb{F}_2^{|\mathsf{Var}(f_{i,j})|}$ be two sets of all configurations of $f_{i,j}$ that are covered by at least one configuration of f_i, f_j, respectively. The two symbols S_i and S_j agree if and only if $L_{i,j} = L_{j,i}$.

When two distinct symbols S_i and S_j do not agree, all configurations in both L_i and L_j that do not cover any configuration of $L_{i,j} \cap L_{j,i}$ are eliminated to obtain an updated set of configurations $L_i' \subseteq L_i, L_j' \subseteq L_j$. The configurations removed from L_i and L_j are indeed the wrong ones because they can not satisfy both f_i and f_j. The set of configurations in $S_i = (\mathsf{Var}(f_i), L_i'), S_j = (\mathsf{Var}(f_j), L_j')$ are updated by L_i', L_j' respectively and both symbol now agree. The step above is referred to as the *agreeing procedure*.

The procedure above is used inside the *Agreeing algorithm*. The algorithm iteratively searches for two distinct polynomials f_i, f_j such that their respective

symbols S_i, S_j disagree and apply the *agreeing procedure* on S_i and S_j. Let f_k be another polynomial such that $\mathsf{Var}(f_j) \cap \mathsf{Var}(f_k)$ is nonempty. Suppose that S_j and S_k disagree when S_i and S_j agree. Applying *agreeing procedure* on S_j, S_k may cause S_i and S_j to disagree. This means that running *agreeing procedure* on one pair can cause other pairs to disagree.

Once all pairs of symbols agree, it is possible that the correct solution is not yet recoverable from each set of configurations. In this case, there are two methods proposed in [RS06] so that its possible to re-start the *Agreeing algorithm*. The two methods are *Splitting* and *Gluing*.

Splitting After all possible pairs of symbols are already in agreeing state without leading to an obvious solution for the system of equations, the splitting strategy chooses a particular set of configurations L_i of a symbol S_i and splits L_i into two parts, $L_i^{(1)}$ and $L_i^{(2)}$. Assuming there is an unique solution, then this is either contained in configuration $L_i^{(i)}$ or in configuration $L_i^{(2)}$. Replacing L_i by $L_i^{(1)}$ or $L_i^{(2)}$ and restarting the *Agreeing algorithm*, the splitting strategy essentially tries to guess which subset matches the unique solution. If a particular choice of subset does not contain the correct configuration, then we proceed by running the *Agreeing algorithm* with the other subset.

It is often the case that performing splitting once is still insufficient to recover a solution for the system of equations, i.e., all distinct pairs of symbols end up in agreeing state again without an obvious solution. What Raddum-Semaev algorithm does to handle this situation is to perform another guess followed by the *Agreeing algorithm* again.

When a wrong subset is chosen from L_i, the *Agreeing algorithm* eventually removes the correct configuration from a set of configurations of a symbol. This means that the *Agreeing algorithm* eliminates all configurations in some symbol S_j. When this event happens, the algorithm backtracks to the last guess made and tries the other subset of configurations.

The splitting strategy can thus be viewed as the traversal of a binary search tree. The leave nodes represent a situation when a solution is found or a particular set of configurations L_i is empty. Note that the depth of the leave nodes varies and the complexity of this approach is exponential in the average of their depth.

Gluing This method can be viewed as the opposite of splitting. It merges two sets of configurations into one, essentially combining two symbols. Let $X_{i,j} = \mathsf{Var}(f_i) \cap \mathsf{Var}(f_j)$ and $Y = \mathsf{Var}(f_i) \cup \mathsf{Var}(f_j)$ where f_i and f_j are two distinct polynomials. We define a set L of configurations for Y that consists of all configurations (p, o, q) such that o is a configuration for $X_{i,j}$, $(p, o) \in L_i$, and $(o, q) \in L_j$. Equivalently, every configuration in L simultaneously covers one configuration of f_i and one configuration of f_j.

Let $S = (Y, L)$ be the symbol constructed after gluing symbols S_i and S_j. The symbols S_i and S_j can then be removed since their respective set of configurations are already contained in L. We can then start the *Agreeing algorithm* again.

The cost of gluing two symbols is that the cardinality of the set of merged configurations is larger. Assume that S_i and S_j are already in agreeing

state. The new set of configurations L of S can have cardinality as small as $\max\{|L_i|, |L_j|\}$ and it can be as large as $|L_i| \cdot |L_j|$.

This approach was generalized later by the same authors in [RS08]. The generalization does not consider multivariate polynomials over $\mathbb{F}_2$ and their configurations but it treats each polynomial as multiple right hand side linear (MRHS) equations.

2.5.5 Mixed Integer-Linear Programming

Another approach to solve a system of polynomial equations over $\mathbb{F}_2$ is by expressing it as a mixed integer-linear programming (MILP) problem. This technique was proposed in [BKS09] to solve polynomial equations of Bivium [Rad06], a small-scale variant of the stream cipher Trivium [Can06].

Definition 2.10 (Mixed-Integer Linear Programming Problem)**.** *Let $k, \ell \in \mathbb{Z}_{\geq 0}$ and set $n = k + \ell$. A Mixed-Integer Linear Programming (MILP) problem is defined as determining*

$$\min_{\mathbf{x}=(\mathbf{x}_1,\ldots,\mathbf{x}_n)} \{c \cdot \mathbf{x} : A\mathbf{x} \leq b, \mathbf{x} \in \mathbb{Z}^k \times \mathbb{R}^\ell\}$$

given a vector $c = (c_1, \ldots, c_n)$, an $m \times n$ matrix A, and a vector $b = (b_1, \ldots, b_m)$. Equivalently, it is a problem of minimizing the linear equation $\sum_{i=1}^{n} c_i \mathbf{x}_i$ subject to the linear inequality constraints defined by A and b. An element $\mathbf{x} \in \mathbb{Z}^k \times \mathbb{R}^\ell$ that satisfies $A\mathbf{x} \leq b$ is called a feasible point. The set of all feasible points is called the feasible set.

In [BKS09], the authors mentioned three (3) techniques to convert a system of multivariate polynomial equations over $\mathbb{F}_2$ to a set of (non-)linear (in-)equality constraints.

Standard Conversion Method. This method is a straightforward approach to express a polynomial over $\mathbb{F}_2$ into equalities in $\mathbb{R}$. It converts binary operations in $\mathbb{F}_2$ in the following way

$$x \cdot y \mapsto \mathbf{x} \cdot \mathbf{y}$$
$$x + y \mapsto \mathbf{x} + \mathbf{y} - 2\mathbf{xy}$$

where x, y and $\mathbf{x}, \mathbf{y}$ are variables over $\mathbb{F}_2$ and $\mathbb{R}$ respectively.

Adapted Standard Conversion (ASC) Method. This second approach takes a polynomial over $\mathbb{F}_2$ and directly converts it into a set of (in)equalities, instead of considering each binary operation in the polynomial. The addition and multiplication in $\mathbb{F}_2$ are converted into addition and multiplication over $\mathbb{R}$. The polynomial over $\mathbb{R}$ is evaluated for all values that holds for the polynomial over $\mathbb{F}_2$. Each of these evaluations yields a new equation over $\mathbb{R}$ by substracting the evaluation result from the left-hand side of the polynomial. It implies that only one of these new polynomials over $\mathbb{R}$ can hold and this is expressed by multiplying each equation with an exclusion variable, followed by adding a new constraint that the sum of all exclusion variables is equal to 1. The following example demonstrates the above steps.

Example 2.11. *Consider the following equation over* $\mathbb{F}_2$ *in* 6 *variables*

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = x_1 + x_2 + x_3x_4 + x_5 + x_6. \tag{2.4}$$

Let $\mathbf{f}$ *be the corresponding polynomial of* f *over* $\mathbb{R}$*. By evaluating* $\mathbf{f}$ *at all solutions of* f*, we get* $0, 2, 4$ *as results. This implies that one of the following linear equations over* $\mathbb{R}$ *is true*

$$\begin{aligned}
\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 &= 0,\\
\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 2 &= 0,\\
\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 4 &= 0.
\end{aligned}$$

Three exclusion variables $\mathbf{x}_{e_1}, \mathbf{x}_{e_2}, \mathbf{x}_{e_3}$ *are then introduced for each equation above.*

$$\begin{aligned}
\mathbf{x}_{e_1}(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6) &= 0,\\
\mathbf{x}_{e_2}(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 2) &= 0,\\
\mathbf{x}_{e_3}(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 4) &= 0,\\
\mathbf{x}_{e_1} + \mathbf{x}_{e_2} + \mathbf{x}_{e_3} &= 1.
\end{aligned}$$

Integer Adapted Standard Conversion (IASC) Method. This conversion technique is an improvement over the ASC method. It follows similar steps as in ASC, except that there is no new equation generated over $\mathbb{R}$. A critical observation used for this method is that when a polynomial over $\mathbb{R}$ is evaluated for all solutions of its corresponding Boolean polynomial, the results are multiples of 2. Instead of generating new equations and adding exclusion variables, this method introduces only one new integer-valued variable. The following example illustrates the benefit of IASC method over ASC.

Example 2.12. *We use the same polynomial as in* (2.4)*. Recall that the results of evaluating* $\mathbf{f}$ *at all solutions of* f *is* $0, 2, 4$*. Thus, a solution of* (2.4) *is a solution to the following equation*

$$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\mathbf{x}_4 + \mathbf{x}_5 + \mathbf{x}_6 - 2y = 0.$$

where $y \in \{0, 1, 2\}$*. The advantage of IASC over ASC is obvious: the degree of the converted polynomial is equal to the original polynomial and the conversion introduces only one new monomial.*

Note that IASC conversion is only applicable if the newly introduced variable can be restricted to an integer value.

The above techniques to generate equalities from an instance of MQ problem over $\mathbb{F}_2$ do not automatically produce an instance of MILP problem, since monomials of degree ≥ 2 still appear in the set of equalities. The final step of the conversion resolves this situation by linearizing the equalities. It substitutes monomials of the form $\mathbf{x}_i\mathbf{x}_j$ by auxiliary variables $\mathbf{y}_{i,j}$. The value $\mathbf{y}_{i,j}$ is equal to zero whenever $\mathbf{x}_i = 0$ or $\mathbf{x}_j = 0$ and this is expressed by the following inequalities

$$\mathbf{y}_{i,j} \leq \mathbf{x}_i, \qquad \mathbf{y}_{i,j} \leq \mathbf{x}_j.$$

The other case where $\mathbf{y}_{i,j} = 1$ if and only if $\mathbf{x}_i = 1$ and $\mathbf{x}_j = 1$ is ensured by the following inequality

$$\mathbf{x}_i + \mathbf{x}_j - 1 \leq \mathbf{y}_{i,j}.$$

In practice, the reduction of a MQ problem over $\mathbb{F}_2$ to a MILP problem combines all three conversion techniques [BKS09]. Once constructed, an off-the-shelf MILP solver such as Gurobi [Gur16], GLPK [Mak18] or SCIP [GEG+17] is used to obtain a feasible solution for the set of linear constraints.

2.5.6 Fast Exhaustive Search

If an instance of an MQ problem in n variables and m equations is defined over a small finite field, a possible approach to find a solution is to perform exhaustive search. When evaluating the system of equations on a possible solution, one can employ an early abort technique that terminates the evaluation after an equation does not evaluate to zero [Nin17].

An improved exhaustive search technique leveraging partial derivatives and Gray code enumeration was proposed in [BCC+10]. The algorithm iterates through all possible solutions in $\mathbb{F}_2^n$ with $\mathcal{O}(\log_2 n \cdot 2^{n+2})$ elementary bit operations. The main idea is that the value of a function f evaluated at $a \in \mathbb{F}_2^n$ can be computed from the value of f and its derivative $D_{e_i} f(x) = f(x + e_i) + f(x)$ evaluated at another point $a' = a + e_i$ where e_i is the i-th canonical basis of $\mathbb{F}_2^n$, that is

$$f(a) = f(a') + D_{e_i} f(a').$$

This can then be applied recursively on the first order derivative. When applying it to a quadratic equation, the first order derivative becomes linear, and the second order derivative becomes constant. The latter serves as the base case for the recursion.

Gray code enumeration ensures that two consecutive elements in $\mathbb{F}_2^n$ differ by only one bit, allowing efficient incremental evaluation. A key optimization in fast-exhaustive search is that not all equations need to be evaluated simultaneously. Instead, only a subset of equations are evaluated with Gray code enumeration and the rest are evaluated using the naive approach when a solution candidate is found. Consequently, the complexity of the algorithm is independent of the number of equations. The implementation in [BCC+10] evaluates a subset of 32 equations with Gray code enumeration in order to utilize the 32-bit registers provided by the hardware.

2.5.7 Other Algorithms

In addition to the methods discussed earlier, several other algorithms exist for solving systems of polynomial equations over finite fields. Among these are probabilistic algorithms for systems over $\mathbb{F}_2$, such as those proposed by Lokshtanov et al. [LPT+17] and Björklund et al [BKW19]. A common feature of both algorithms is that they asymptotically outperform exhaustive search algorithms in the worst case and their complexity does not depend on any assumption about the original system of equations. Other approaches combine partial exhaustive search with algebraic solving procedure such as BooleanSolve [BFSS13], FXL [CKPS00], and Hybrid F_4/F_5 [BFP09]. Moreover, several algorithms have been specifically designed to solve underdefined system of equations [KPG99, MHT13, CGMT02], where the number of variables exceeds the

number of equations. For a comprehensive overview of existing techniques for solving systems of equations over finite fields, we refer the reader to the survey by Bellini et al. [BMSV22] which provides a detailed discussion of their computational complexity.

3 Theory of Gröbner Bases

Contents

This chapter is an overview of mathematical theory on Gröbner bases. The notion of Gröbner basis for a polynomial ideal was introduced in 1965 by Bruno Buchberger in his PhD thesis [Buc65, Buc06].* Buchberger also proposed an algorithm that computes a Gröbner basis given a basis or a set of generators for an ideal. In this chapter, we limit ourselves to describing Gröbner bases as mathematical objects and their applications. The explanation on algorithms that compute Gröbner bases shall be given in Chapter 4

This chapter is divided into five sections. We first introduce the notations, definitions, and necessary preliminaries in Section 3.1. This will be followed by the definition of monomial orderings and their examples in Section 3.2. Section 3.3 introduces the division algorithm in a multivariate polynomial ring, generalizing the existing algorithm in the univariate case. A natural characterization of the remainder together with the dedicated algorithm that computes it are also given in the same section. The definition of Gröbner bases and some of its properties will be presented in Section 3.4. Two relevant applications of Gröbner bases in this thesis are going to be described in Section 3.5. This chapter is a collection of topics from various existing literature on Gröbner bases, such as Chapter 1-3 of [CLO15] and [BW93, KR00].

3.1 Notations and Preliminaries

Definition 3.1 (Group)**.** *A group is a set G together with a binary operation $* : G \times G \mapsto G$ such that*

1. *The binary operation $*$ is associative, that is $a * (b * c) = (a * b) * c$ for any $a, b, c \in G$.*
2. *There exists an element $0 \in G$, called the identity, such that $0 * a = a * 0 = a$ for all $a \in G$.*
3. *For every $a \in G$, there exists an element $b \in G$ such that $a * b = b * a = 0$.*

If the group G also satisfies $a * b = b * a$ for all $a, b \in G$, then G is said to be Abelian.

Definition 3.2 (Subgroup)**.** *A subset H of a group G is a subgroup of G if H is itself a group w.r.t. the operation of G.*

* The name "Gröbner bases" is originated from Buchberger's PhD advisor, Wolfgang Gröbner.

Definition 3.3 (Ring)**.** *A ring is a set R together with two binary operations $+, \cdot : R \times R \mapsto R$, called addition and multiplication respectively, such that*

1. *R is an Abelian group under $+$.*
2. *The binary operation $\cdot$ is associative, that is $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for any $a, b, c \in R$.*
3. *The distributive laws hold, that is, for all $a, b, c \in R$ we have $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$.*

A ring R is called a ring with identity if there exists an element $1 \in R$ such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in R$. A ring R is called a commutative ring if $a \cdot b = b \cdot a$ for all $a, b \in R$.

Definition 3.4 (Characteristic of a ring)**.** *Let R be a ring. If there exists a positive integer n such that $nr = 0$ for all $r \in R$, then the least such positive integer is called the characteristic of R. If no such positive integer exists, then R is of characteristic 0.*

Definition 3.5 (Field)**.** *A ring $\{0\} \neq R$ is called a field if R is a commutative ring with identity and for all $0 \neq a \in R$ there exists an element $b \in R$ such that $a \cdot b = 1$. In other words, the nonzero elements of R form a group w.r.t. the multiplication.*

Definition 3.6 (Ideal)**.** *A subset $I \subseteq R$ of a commutative ring R is an ideal of R if it satisfies:*

1. *$0 \in I$,*
2. *If $f, g \in I$ then $f + g \in I$,*
3. *If $f \in I$ and $h \in R$ then $hf = fh \in I$.*

Remark. *The notion of ideal also exists for noncommutative ring. That is, a left (resp. right) ideal is a subset I of a ring R which is an additive subgroup of R and such that for all $f \in I$ and all $h \in R$ we have $hf \in I$ (resp. $fh \in I$).*

We use $\mathbb{Z}$ to denote the ring of integers and $\mathbb{Z}_{\geq 0}$ as the set of nonnegative integers. The symbols $\mathbb{F}$ and $\mathbb{F}_q$ are used to denote an arbitrary field and the finite field of order q, respectively.

The central subject of this thesis deals with polynomials in finitely many variables. We start by introducing monomials, being the basic building blocks of a polynomial, together with some of its related definitions below.

Definition 3.7. *A **monomial** in variables $x_1, \ldots, x_n$ is a product of the form*

$$x_1^{\alpha_1} \cdots x_n^{\alpha_n}$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_{\geq 0}$. The notation for monomial above can be simplified as x^α where $x = (x_1, \ldots, x_n)$ and $\alpha = (\alpha_1, \ldots, \alpha_n)$. For the case $\alpha = (0, \ldots, 0)$ we simply write x^α as 1.

Definition 3.8. *Let $\alpha, \beta \in \mathbb{Z}_{\geq 0}^n$ with $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$. We say that the monomial x^α **divides** x^β, denoted by $x^\alpha \mid x^\beta$, if $\alpha_i \leq \beta_i$ for all $i \in \{1, \ldots, n\}$.*

Definition 3.9. *A **polynomial** in n variables $x_1, \ldots, x_n$ with coefficients in a field $\mathbb{F}$ is a finite linear combination (with coefficients in $\mathbb{F}$) of monomials. We shall write a polynomial f in the form*

$$f = \sum_{\alpha \in \mathcal{A}} \mathfrak{c}_\alpha x^\alpha, \quad \mathfrak{c}_\alpha \in \mathbb{F}$$

where $\mathcal{A}$ is a finite subset of $\mathbb{Z}^n_{\geq 0}$. If $\mathcal{A}$ is empty or $\mathfrak{c}_\alpha = 0$ for all $\alpha \in \mathcal{A}$, we shall write $f = 0$ and f is said to be the zero polynomial.

Definition 3.10. *The set of all polynomials in variables $x_1, \ldots, x_n$ with coefficients in the field $\mathbb{F}$ is denoted by*

$$\mathbb{F}[x_1, \ldots, x_n].$$

The set $\mathbb{F}[x_1, \ldots, x_n]$ is a commutative ring with identity with respect to polynomial addition and multiplication.

Notation. *Throughout this thesis, the polynomial ring $\mathbb{F}[x_1, \ldots, x_n]$ will be denoted by $\mathcal{R}$ unless explicitly stated otherwise.*

Remark 3.11. *The ring $\mathcal{R}$ can also be viewed as an infinite-dimensional vector space over $\mathbb{F}$ and the set of all monomials in $\mathcal{R}$ serves as a basis of $\mathcal{R}$.*

Definition 3.12. *Let $f = \sum_\alpha \mathfrak{c}_\alpha x^\alpha$ be a nonzero polynomial in $\mathcal{R}$.*

1. *If $\mathfrak{c}_\alpha \neq 0$ then we call $\mathfrak{c}_\alpha x^\alpha$ a **term** of f.* We say that a term $\mathfrak{t}$ divides another term $\mathfrak{t}'$, denoted by $\mathfrak{t} \mid \mathfrak{t}'$, if the monomial of $\mathfrak{t}$ divides the monomial of $\mathfrak{t}'$.*

2. *For a term $\mathfrak{c}_\alpha x^\alpha$ of f, we call $\mathfrak{c}_\alpha$ the **coefficient** of the monomial x^α.*

3. *The set of all terms of f and the set of all monomials of f are respectively denoted by*

$$\mathsf{T}(f) = \{\mathfrak{c}_\alpha x^\alpha : \mathfrak{c}_\alpha \neq 0\},$$
$$\mathsf{M}(f) = \{x^\alpha : \mathfrak{c}_\alpha \neq 0\}.$$

4. *The definition of $\mathsf{T}(f), \mathsf{M}(f)$ are naturally extended for a subset of polynomials $F \subseteq \mathcal{R}$, that is*

$$\mathsf{T}(F) = \bigcup_{f \in F} \mathsf{T}(f),$$
$$\mathsf{M}(F) = \bigcup_{f \in F} \mathsf{M}(f).$$

   The set of all terms in $\mathcal{R}$ and all monomials in $\mathcal{R}$ are then denoted by $\mathsf{T}(\mathcal{R}), \mathsf{M}(\mathcal{R})$ respectively.

* In other existing literature, some authors interchange our definition of term and monomial such as in [Fau99, BW93].

5. *The degree of a nonzero $f \in \mathcal{R}$ is defined as*

$$\deg(f) = \max_{\alpha} \left\{ \sum_{i=1}^{n} \alpha_i : \mathfrak{c}_\alpha \neq 0 \right\}.$$

The degree of the zero polynomial is equal to $-\infty$ by convention.

Definition 3.13. *Let $F = \{f_1, \ldots, f_m\} \subseteq \mathcal{R}$. We define the following set*

$$\langle F \rangle = \langle f_1, \ldots, f_m \rangle = \left\{ \sum_{i=1}^{m} h_i f_i : h_1, \ldots, h_m \in \mathcal{R} \right\}.$$

*The set $\langle F \rangle = \langle f_1, \ldots, f_m \rangle$ is an ideal of $\mathcal{R}$. We call $\langle f_1, \ldots, f_m \rangle$ the **ideal generated by** $f_1, \ldots, f_m$.*

3.2 Monomial Orderings

The first question that arises when dealing with multivariate polynomials is how the monomials are supposed to be ordered. While it is trivial for the case of univariate polynomials, the situation is no longer obvious when more variables involved. For this purpose one needs to define the notion of *monomial ordering*, that allows unambiguous arrangement of terms in a polynomial and at the same time preserves the compatibility with the algebraic structure of multivariate polynomial rings.

Definition 3.14 (Monomial Ordering [CLO15])**.** *A monomial ordering $\geq$ is a relation on $\mathsf{M}(\mathcal{R})$ satisfying:*

1. *$\geq$ is a total ordering on $\mathsf{M}(\mathcal{R})$, i.e., it satisfies the following*
   (a) *$x^\alpha \geq x^\alpha$ for all $x^\alpha \in \mathsf{M}(\mathcal{R})$.*
   (b) *$x^\alpha \geq x^\beta$ and $x^\beta \geq x^\alpha$ implies $x^\alpha = x^\beta$.*
   (c) *$x^\alpha \geq x^\beta$ and $x^\beta \geq x^\gamma$ implies $x^\alpha \geq x^\gamma$.*
   (d) *For any $x^\alpha, x^\beta \in \mathsf{M}(\mathcal{R})$, either $x^\alpha \geq x^\beta$ or $x^\beta \geq x^\alpha$*
2. *If $x^\alpha \geq x^\beta$ and $x^\gamma \in \mathsf{M}(\mathcal{R})$, then $x^\alpha x^\gamma \geq x^\beta x^\gamma$.*
3. *$\geq$ is a well-ordering on $\mathsf{M}(\mathcal{R})$, that is every nonempty subset of $\mathsf{M}(\mathcal{R})$ has a smallest element under $\geq$.*

Remark 3.15. *Since any monomial $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ can be associated with the n-tuple of exponents $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}^n_{\geq 0}$ and vice versa, it follows that a monomial ordering is also an ordering on the set $\mathbb{Z}^n_{\geq 0}$.*

Proposition 3.16. *An ordering on $\mathsf{M}(\mathcal{R})$ is a well-ordering if and only if every strictly decreasing sequence in $\mathsf{M}(\mathcal{R})$*

$$\mathfrak{m}_1 > \mathfrak{m}_2 > \mathfrak{m}_3 > \cdots$$

eventually terminates

Proof. See the proof of Lemma 2 in [CLO15, pg. 56]. ■

We mention three examples of monomial ordering below: lexicographic (*lex*), degree lexicographic (*deglex*), and degree-reverse lexicographic ordering (*degrevlex*). Other monomial orderings do exist, however these are the most relevant orderings within the scope of this thesis.

Definition 3.17 (Lexicographic Order)**.** *Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ be elements of $\mathbb{Z}^n_{\geq 0}$. We say that $x^\alpha >_{\text{lex}} x^\beta$ if and only if the leftmost nonzero entry of $\alpha - \beta \in \mathbb{Z}^n$ is positive.*

Definition 3.18 (Degree Lexicographic Ordering)**.** *Let $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$. We say that $x^\alpha >_{\text{deglex}} x^\beta$ if and only if:*

- $\deg(x^\alpha) > \deg(x^\beta)$ *or*
- $\deg(x^\alpha) = \deg(x^\beta)$ *and* $x^\alpha >_{\text{lex}} x^\beta$.

Definition 3.19 (Degree-Reverse Lexicographic Ordering)**.** *Let $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$. We say that $x^\alpha >_{\text{degrevlex}} x^\beta$ if and only if:*

- $\deg(x^\alpha) > \deg(x^\beta)$ *or*
- $\deg(x^\alpha) = \deg(x^\beta)$ *and the rightmost nonzero entry of $\alpha - \beta \in \mathbb{Z}^n$ is negative.*

In addition to *lex* and *degrevlex*, we also mention a generic ordering below called *block ordering*, which allows the incorporation of multiple monomial orderings for disjoint subsets of the variables.

Definition 3.20 (Block Ordering)**.** *Let $(x_1, \ldots, x_i)$ and $(y_1, \ldots, y_j)$ be two ordered sets of variables. Let $>_1, >_2$ be monomial orderings on $\mathbb{F}[x_1, \ldots, x_i]$ and $\mathbb{F}[y_1, \ldots, y_j]$, respectively. We say that $x^\alpha y^\beta > x^A y^B$ with respect to the block ordering $(>_1, >_2)$ on $\mathbb{F}[x_1, \ldots, x_i, y_1, \ldots, y_j]$ if and only if:*

- $x^\alpha >_1 x^A$ *or*
- $x^\alpha = x^A$ *and* $y^\beta >_2 y^B$.

Remark. *From now on whenever a polynomial ring $\mathcal{R}$ is defined, we adopt a monomial ordering $\geq$ on $\mathsf{M}(\mathcal{R})$.*

Definition 3.21 (Monomial Ideal [CLO15])**.** *An ideal $I \subseteq \mathcal{R}$ is a **monomial ideal** if there exists a (possibly infinite) subset $A \subseteq \mathsf{M}(\mathcal{R})$ such that $I = \langle A \rangle$.*

Proposition 3.22. *Let $A \subseteq \mathsf{M}(\mathcal{R})$ and let $I = \langle A \rangle$ be a monomial ideal. A monomial $\mathfrak{m} \in \mathsf{M}(\mathcal{R})$ is an element of I if and only if $\mathfrak{m}$ is divisible by a monomial $\mathfrak{m}'$ for some $\mathfrak{m}' \in A$.*

Proof. See the proof of Lemma 2 in [CLO15, pg. 70]. ■

Theorem 3.23 (Dickson's Lemma)**.** *Let $A \subseteq \mathsf{M}(\mathcal{R})$ and let $I = \langle A \rangle$ be a monomial ideal. Then I can be written in the form $I = \langle \mathfrak{m}_1, \ldots, \mathfrak{m}_s \rangle$ for some $\mathfrak{m}_1, \ldots, \mathfrak{m}_s \in A$. Thus, being finitely generated by monomials, we say that the ideal I has a finite basis.*

Proof. See the proof of Theorem 5 in [CLO15, pg. 72]. ■

Once the notion of monomial ordering is established, we define the following terminologies related to the largest monomial of nonzero polynomials in $\mathcal{R}$.

Definition 3.24. *Let $f = \sum_\alpha \mathfrak{c}_\alpha x^\alpha$ be a nonzero polynomial in $\mathcal{R}$ and let $\geq$ be a monomial ordering.*

1. *The **multidegree** of f is*

$$\text{multdeg}(f) = \max\{\alpha \in \mathbb{Z}^n_{\geq 0} : \mathfrak{c}_\alpha \neq 0\}$$

   where the maximum is taken with respect to $\geq$.

2. *The **leading coefficient** of f is $\mathsf{LC}(f) = \mathfrak{c}_{\text{multdeg}(f)} \in \mathbb{F}$.*

3. *The **leading monomial** of f is $\mathsf{LM}(f) = x^{\text{multdeg}(f)}$.*

4. *The **leading term** of f is $\mathsf{LT}(f) = \mathsf{LC}(f) \cdot \mathsf{LM}(f)$.*

5. *The **tail*** of f is $\mathsf{Tail}(f) = f - \mathsf{LT}(f)$.*

3.3 Polynomial Reduction

One of the important algorithms to work with multivariate polynomials is the computation of the remainder of $f \in \mathcal{R}$ after division by a finite nonempty ordered subset $G \subseteq \mathcal{R}$. The central idea of the algorithm relies on the existence of division algorithms in $\mathcal{R}$.

Theorem 3.25 ([CLO15, KR00])**.** *Let $G = \{g_1, \ldots, g_s\}$ be a nonempty finite subset of $\mathcal{R}$. Then every polynomial $f \in \mathcal{R}$ can be written as*

$$f = \sum_{i=1}^{s} q_i g_i + r, \tag{3.1}$$

where $q_1, \ldots, q_s, r \in \mathcal{R}$ such that

1. *either $r = 0$ or no monomials of r is divisible by any of $\mathsf{LM}(g_1), \ldots, \mathsf{LM}(g_s)$,*

2. *if $q_i g_i \neq 0$ then $\mathsf{LM}(q_i g_i) \leq \mathsf{LM}(f)$.*

Proof. The pseudo-code of the division algorithm is given in Algorithm 3.1 for the computation of r and $q_1, \ldots, q_s$. Starting with $p = f$, in every step of the iteration, the algorithm tries to eliminate the leading term of p, using polynomials in G whenever possible. During the execution of the algorithm, either one of the two events below happens :

1. When some $\mathsf{LT}(g_i)$ divides $\mathsf{LT}(p)$, we call this step the *division step*. This is seen in the line 6-9.

2. If there exists no $\mathsf{LT}(g_i)$ that divides $\mathsf{LT}(p)$, we call this step the *remainder step*. This is seen in the line 11-12.

* Also called reductum in [BW93].

Termination : Let p_j denote the value of p after the j-th iteration where $p_0 = f$. Suppose that at the j-th iteration, the division step occurred. We have $p_j = p_{j-1} - \mathsf{LT}(p_{j-1})/\mathsf{LT}(g_i) \cdot g_i$ and

$$\begin{aligned}\mathsf{LT}(\mathsf{LT}(p_{j-1})/\mathsf{LT}(g_i) \cdot g_i) &= \mathsf{LT}(p_{j-1})/\mathsf{LT}(g_i) \cdot \mathsf{LT}(g_i) \quad \text{(by 2 of Definition 3.14)} \\ &= \mathsf{LT}(p_{j-1}).\end{aligned}$$

so that p_{j-1} and $\mathsf{LT}(p_{j-1})/\mathsf{LT}(g_i) \cdot g_i$ have the same leading term. This implies that $\mathsf{LM}(p_j) < \mathsf{LM}(p_{j-1})$ when $p_j \neq 0$. Next, suppose that at the j-th iteration, the remainder step occurred. We have $p_j = \mathsf{Tail}(p_{j-1}) = p_{j-1} - \mathsf{LT}(p_{j-1})$. Here, it is obvious that $\mathsf{LM}(p_j) < \mathsf{LM}(p_{j-1})$ when $p_j \neq 0$. Thus, in either case we have

$$\mathsf{LM}(p_0) > \mathsf{LM}(p_1) > \mathsf{LM}(p_2) > \cdots .$$

If the algorithm never terminated, then we would get an infinite decreasing sequence of leading monomials. However, this contradicts the well-ordering property of $>$ in Proposition 3.16. Eventually, $p = 0$ must happen so that the algorithm terminates after finitely many iterations.

Correctness : To prove the correctness, we will show that

$$f = \sum_{i=1}^{s} q_i g_i + p + r \tag{3.2}$$

holds at every iteration. This is clearly true during the initialization since $q_1 = \ldots = q_s = r = 0$ and $p = f$. Suppose that (3.2) holds at one step of the while-loop. If the algorithm executed the division step, then there exists $g_i \in G$ such that $\mathsf{LT}(g_i) \mid \mathsf{LT}(p)$ and the following equality

$$q_i g_i + p = (q_i + \mathsf{LT}(p)/\mathsf{LT}(g_i))g_i + (p - \mathsf{LT}(p)/\mathsf{LT}(g_i) \cdot g_i)$$

shows that $q_i g_i + p$ remains unaffected. Since other variables are also unchanged, then (3.2) remains true in this case. Now if, instead of the division step, the algorithm executed the remainder step then it affects both p and r. However, the sum $p + r$ is unchanged since

$$p + r = \mathsf{Tail}(p) + (r + \mathsf{LT}(p))$$

and therefore (3.2) remains true.

Note that the algorithm is terminated when $p = 0$ which (3.2) eventually becomes

$$f = \sum_{i=1}^{s} q_i g_i + r.$$

Since the leading term of p in each iteration is added to r whenever it is not divisible by any of $\mathsf{LT}(g_1), \ldots, \mathsf{LT}(g_s)$ it follows that r has the desired property when the algorithm terminates.

It remains to show the relation between $\mathsf{LM}(f)$ and $\mathsf{LM}(q_i g_i)$ when $q_i g_i \neq 0$. Every term of q_i is of the form $\mathsf{LT}(p)/\mathsf{LT}(g_i)$ for some polynomial $0 \neq p$. Initially $p = f$ and in the proof of termination we showed that the leading monomial of p decreases in every iteration. This implies that that $\mathsf{LM}(p) \leq \mathsf{LM}(f)$. Using 2 of Definition 3.14, it follows that $\mathsf{LM}(q_i g_i) \leq \mathsf{LM}(f)$ when $q_i g_i \neq 0$.

■

Input: $G = \{g_1, \ldots, g_s\} \subseteq \mathcal{R}, f \in \mathcal{R}$
Output: $q_1, \ldots, q_s, r \in \mathcal{R}$ such that $f = \sum_{i=1}^{s} q_i g_i + r$ and either $r = 0$ or no monomial of r is divisible by any of $\mathsf{LM}(g_1), \ldots, \mathsf{LM}(g_s)$

```
1  (q_1, ..., q_s) ← (0, ..., 0)
2  r ← 0
3  p ← f
4  while p ≠ 0 do
5      if ∃ g_i ∈ G : LT(g_i) | LT(p) then
6          Select g_i ∈ G such that LT(g_i) | LT(p)
7          t ← LT(p)/LT(g_i)
8          q_i ← q_i + t
9          p ← p − t · g_i
10     else
11         r ← r + LT(p)
12         p ← Tail(p)
13 return q_1, ..., q_s, r
```

Algorithm 3.1: The division algorithm in $\mathcal{R}$.

Remark 3.26. *The description of Algorithm 3.1 leaves some degrees of freedom in the selection of g_i in line 6. One way is to treat G as an ordered set $(g_1, \ldots, g_s)$ and choose the smallest $i \in \{1, \ldots, s\}$ such that $\mathsf{LT}(g_i) \mid \mathsf{LT}(p)$. This strategy is used in the description of the division algorithm in [CLO15, pg. 64]. In that respect we denote by $\overline{f}^G$ the remainder after division of f by an ordered set $G \subseteq \mathcal{R}$.*

Definition 3.27. *Let $f \in \mathcal{R}$ be a nonzero polynomial. A polynomial $0 \neq g \in \mathcal{R}$ is a **reductor** of f if $\mathsf{LM}(g) \in \mathsf{M}(f)$.*

Definition 3.28. *An algorithm that computes the remainder r (by omitting the quotients q_i) in Theorem 3.25 is called a **reduction algorithm**.*

In this work we consider a family of reduction algorithms, instead of one dedicated algorithm, such that each of the algorithms eliminates terms appearing in a specific part of a polynomial. Before describing them, we first introduce the notion of "reducibility".

Definition 3.29. *Let $0 \neq f \in \mathcal{R}$ and $G = \{g_1, \ldots, g_t\}$ be a subset of $\mathcal{R}$.*

1. *The polynomial f is **reducible** by G if there exists a monomial $\mathfrak{m} \in \mathsf{M}(f)$ such that $\mathfrak{m} \in \langle \mathsf{LM}(g_1), \ldots, \mathsf{LM}(g_t) \rangle$.*
2. *The polynomial f is **lead-reducible** by G if $\mathsf{LM}(f) \in \langle \mathsf{LM}(g_1), \ldots, \mathsf{LM}(g_t) \rangle$.*
3. *The polynomial f is **tail-reducible** by G if $\mathsf{Tail}(f) \neq 0$ and $\mathsf{Tail}(f)$ is reducible by G.*

Definition 3.29 suggests formulation of reduction algorithms that are applied on terms appearing in the leading term of a polynomial, the polynomial itself, and its tail. We describe three reduction algorithms below.

1. LeadReduce: This reduction algorithm plays the role of iteratively cancelling the leading term of a polynomial $f \in \mathcal{R}$ by an ordered set $G \subset \mathcal{R}$ until f is no longer lead-reducible by G. The description is given in Algorithm 3.2.

2. FullReduce: The FullReduce algorithm returns either a zero polynomial or a polynomial that is not reducible by G. One can verify this is equal to computing the remainder of f after division by G. The iterative reduction on f is done by calling the LeadReduce algorithm. This is formally described in Algorithm 3.3.

3. TailReduce: The TailReduce algorithm returns a nonzero polynomial that is not tail-reducible by G. It applies FullReduce algorithm on the tail of f. The pseudo-code is given in Algorithm 3.4.

Input: A polynomial $f \in \mathcal{R}$
Input: An ordered subset $G = (g_1, \ldots, g_t) \subset \mathcal{R}$
Result: A polynomial f such that $f = 0$ or $\mathsf{LM}(f) \notin \langle \mathsf{LM}(g_1), \ldots, \mathsf{LM}(g_t) \rangle$.

```
if f = 0 then
    return f
for i = 1 to t do
    if LT(g_i) | LT(f) then
        t ← LT(f)/LT(g_i)
        return LeadReduce(f − t · g_i, G)
return f
```

Algorithm 3.2: LeadReduce Algorithm.

Input: A polynomial $f \in \mathcal{R}$.
Input: An ordered subset $G = (g_1, \ldots, g_t) \subset \mathcal{R}$.
Result: A polynomial r such that $r = 0$ or for all $\mathfrak{m} \in \mathsf{M}(r)$: $\mathfrak{m} \notin \langle \mathsf{LM}(g_1), \ldots, \mathsf{LM}(g_t) \rangle$.

```
r ← 0
f ← LeadReduce(f, G)
while f ≠ 0 do
    r ← r + LT(f)
    f ← LeadReduce(f − LT(f), G)
return r
```

Algorithm 3.3: FullReduce Algorithm.

3.4 Gröbner Bases

This section presents the definition of Gröbner bases as the central subject of this thesis. Gröbner bases are considered as the "nice" basis for polynomial ideals. Its properties, that will be explained later, help in solving various problems in computational commutative algebra. Before defining Gröbner bases in

Input: A polynomial $f \in \mathcal{R}$.
Input: An ordered subset $G \subseteq \mathcal{R}$.
Result: A polynomial f such that $\mathsf{Tail}(f) = 0$ or
$\mathfrak{m} \notin \langle \mathsf{LM}(g_1), \ldots, \mathsf{LM}(g_t) \rangle$ for all $\mathfrak{m} \in \mathsf{M}(\mathsf{Tail}(f))$.
1 **return** $\mathsf{LT}(f) + \text{FullReduce}(\mathsf{Tail}(f), G)$

Algorithm 3.4: TailReduce Algorithm.

Definition 3.31, we want to recall the Hilbert basis theorem, that describes a fundamental property of polynomial ideals.

Theorem 3.30 (Hilbert Basis Theorem)**.** *Every ideal $I \subset \mathcal{R}$ has a finite generating set, i.e. $I = \langle g_1, \ldots, g_t \rangle$ for some $g_1, \ldots, g_t \in I$.*

Proof. See the proof of Theorem 4 in [CLO15, pg. 77]. ■

Definition 3.31 (Gröbner basis)**.** *Fix a monomial order on $\mathsf{M}(\mathcal{R})$. A finite subset G of an ideal $\{0\} \neq I \subseteq \mathcal{R}$ is said to be a **Gröbner basis** of I if for all $0 \neq f \in I$, there exists a polynomial $g \in G$ such that $\mathsf{LT}(g) \mid \mathsf{LT}(f)$.*

Theorem 3.32. *Every ideal $\{0\} \neq I \subseteq \mathcal{R}$ has a Gröbner basis. Moreover, any Gröbner basis for I is a basis of I.*

Proof. See Corollary 6 in Section §2.5 of [CLO15]. ■

Theorem 3.33 (The Ascending Chain Condition)**.** *Let $I_1 \subseteq I_2 \subseteq I_3 \subseteq \cdots$ be ascending chain of ideals in $\mathbb{F}[x_1, \ldots, x_n]$. Then there exists a positive integer N such that $I_N = I_{N+1} = I_{N+2} = \ldots$.*

Proof. See the proof of Theorem 7 in Section §2.5 of [CLO15, pg. 80]. ■

Remark 3.34. *A commutative ring that satisfies the property in Theorem 3.33 is also called a Noetherian ring.*

We remark that Definition 3.31 allows inclusion of some unnecessary elements in a Gröbner basis. Let G be a Gröbner basis of an ideal I and let $p \in G$. If there exists a polynomial $g \in G \setminus \{p\}$ such that $\mathsf{LT}(g) \mid \mathsf{LT}(p)$, we can remove p from G as $G \setminus \{p\}$ remains a Gröbner basis of I. This observation, together with adjustment of coefficients to normalize the leading coefficient of elements in G leads to the definition of minimal Gröbner bases.

Definition 3.35. *A Gröbner basis G of an ideal I is a **minimal Gröbner basis** if for all $g \in G$*

1. $\mathsf{LC}(g) = 1$ *and*
2. $\mathsf{LM}(g) \notin \langle \mathsf{LM}(G \setminus \{g\}) \rangle$.

In addition to Definition 3.35, there exists a stronger type of minimal Gröbner bases that adds the condition that every polynomial $g \in G$ is not reducible by $G \setminus \{g\}$.

Definition 3.36. *A Gröbner basis G of an ideal I is a **reduced Gröbner basis** if for all $g \in G$*

1. $\mathsf{LC}(g) = 1$ *and*
2. *For all $\mathfrak{m} \in \mathsf{M}(g)$, the monomial $\mathfrak{m} \notin \langle \mathsf{LM}(G \setminus \{g\}) \rangle$.*

3.5 Applications

One of the main reasons behind the extensive study of Gröbner bases for the past decades is its diverse application across numerous disciplines in mathematics. The wide-spectrum of its applications makes it impossible to exhaustively mention all of it. Some of them are already discussed, for example, in Chapter 3 of [KR00] and Chapter 2 of [AL94]. We refer to [SSM+09] for readers who are interested in applications related to coding theory and cryptography.

Within the scope of this thesis, we only describe two well-known applications of Gröbner bases in this section.

3.5.1 Ideal Membership Problem

Problem (Ideal Membership). *Given an ideal $I = \langle f_1, \ldots, f_m \rangle \subseteq \mathcal{R}$, determine whether a given polynomial $f \in \mathcal{R}$ lies in I.*

Although the division algorithm seems to be an obvious solution for this problem, i.e., if the remainder after division of f by the ordered set $G = (f_1, \ldots, f_m)$ is zero then $f \in I$, we want to recall that uniqueness of the remainder and the quotients in the division algorithm relies on the ordering of elements in G. Thus, we may encounter a situation where $f \in I$ and yet the remainder on the division of f by G is nonzero.

Gröbner bases then plays the role of remedies in such situation. The following property of Gröbner bases, together with the division algorithm, solves the ideal membership problem in multivariate polynomial rings.

Proposition 3.37 ([CLO15]). *Let $I \subseteq \mathcal{R}$ be an ideal and let G be a Gröbner basis of I. Given $f \in \mathcal{R}$, there exists a unique $r \in \mathcal{R}$ with the following properties*

1. *For any $g \in G$, no term of r is divisible by $\mathsf{LT}(g)$.*
2. *There exists a $q \in I$ such that $f = q + r$.*

More specifically, r is the remainder after division of f by G, no matter how the elements of G are ordered when using the division algorithm.

Proof. See Proposition 1 in Section §2.6 of [CLO15]. ■

Corollary 3.38 ([CLO15]). *Let $I \subseteq \mathcal{R}$ be an ideal and let $G \subseteq I$ be a Gröbner basis of I. The polynomial $f \in \mathcal{R}$ is an element of I if and only if the remainder after division of f by G is zero.*

Proof. First, it is obvious that if the remainder is zero, it immediately follows that $f \in I$. To prove the converse for any $f \in I$, we can write $f = f + 0$. Since G is a Gröbner basis, $f = f + 0$ satisfies both conditions in Proposition 3.37. Thus, 0 is the remainder after division of f by G. ■

Assume that a Gröbner basis G of a polynomial ideal I is known. We already have an algorithm that determines whether a polynomial $f \in \mathcal{R}$ is an element of I by computing the remainder after division of f by G with the division or FullReduce algorithm. The complete solution for the ideal membership problem requires an algorithm that obtains a Gröbner basis from an arbitrary basis of a polynomial ideal. This will be further described in Chapter 4.

3.5.2 Solving Systems of Polynomial Equations

Another well-known application of Grobner bases, in addition to solving the ideal membership problem, is its ability to solve systems of polynomial equations.

Problem. *Let $f_1, \ldots, f_m \in \mathcal{R}$. Find all solutions $(x_1, \ldots, x_n) \in \mathbb{F}^n$ of the following system of polynomial equations*

$$f_1(x_1, \ldots, x_n) = \ldots = f_m(x_1, \ldots, x_n) = 0.$$

The analogous concept in algebraic geometry for a set that consists of solutions of polynomial equations is known as an affine variety.

Definition 3.39. *Let $f_1, \ldots, f_m$ be polynomials in $\mathcal{R}$. We define the set*

$$\mathbf{V}(f_1, \ldots, f_m) = \{(a_1, \ldots, a_n) \in \mathbb{F}^n : f_i(a_1, \ldots, a_n) = 0,\ \forall i \in \{1, \ldots, m\}\}$$

*and we call $\mathbf{V}(f_1, \ldots, f_m)$ the **affine variety** defined by $f_1, \ldots, f_m$.*

Both propositions below are crucial in order to understand the relation between the affine variety defined by a finite set of polynomials and the corresponding ideal generated by the same polynomials.

Proposition 3.40. *If $f_1, \ldots, f_m \in \mathcal{R}$ and $g_1, \ldots, g_t \in \mathcal{R}$ are bases of the same ideal, i.e. $\langle f_1, \ldots, f_m \rangle = \langle g_1, \ldots, g_t \rangle$, then we have*

$$\mathbf{V}(f_1, \ldots, f_m) = \mathbf{V}(g_1, \ldots, g_t).$$

Proof. Let $(a_1, \ldots, a_n) \in \mathbf{V}(f_1, \ldots f_m)$. Since $\langle f_1, \ldots, f_m \rangle = \langle g_1, \ldots, g_t \rangle$ then $g_1, \ldots, g_t \in \langle f_1, \ldots, f_m \rangle$. This implies that, every g_j can be written as

$$g_j = \sum_{i=i}^{m} h_i f_i, \quad h_i \in \mathcal{R}.$$

This shows that for all $j = 1, \ldots, t$, we have

$$\begin{aligned} g_j(a_1, \ldots, a_n) &= \sum_{i=1}^{m} h_i(a_1, \ldots, a_n) f_i(a_1, \ldots, a_n) \\ &= 0. \end{aligned}$$

Thus $(a_1, \ldots, a_n) \in \mathbf{V}(g_1, \ldots, g_t)$ and hence $\mathbf{V}(f_1, \ldots, f_m) \subseteq \mathbf{V}(g_1, \ldots, g_t)$. The other direction of the proof can be done in a similar manner by interchanging the role of $\langle f_1, \ldots, f_m \rangle$ and $\langle g_1, \ldots, g_t \rangle$, i.e., expressing each f_i as element of $\langle g_1, \ldots, g_t \rangle$. ■

Proposition 3.41. *Let I be an ideal of $\mathcal{R}$ and let $\mathbf{V}(I)$ be the following set*

$$\mathbf{V}(I) = \{(a_1, \ldots, a_n) \in \mathbb{F}^n : f(a_1, \ldots, a_n) = 0,\ \forall f \in I\}.$$

The set $\mathbf{V}(I)$ is an affine variety. In particular, if $I = \langle f_1, \ldots, f_m \rangle$, then $\mathbf{V}(I) = \mathbf{V}(f_1, \ldots, f_m)$.

Proof. By the Hilbert Basis Theorem, there exists a finite generating set of polynomials $\{f_1, \ldots, f_m\}$ for I, i.e. $I = \langle f_1, \ldots, f_m \rangle$. Let $(a_1, \ldots, a_n) \in \mathbf{V}(I)$. Since $f_1, \ldots, f_m \in I$, then $f_i(a_1, \ldots, a_n) = 0$ for all $i = 1, \ldots, m$ by definition and this shows that $(a_1, \ldots, a_n) \in \mathbf{V}(f_1, \ldots, f_m)$. Hence $\mathbf{V}(I) \subseteq \mathbf{V}(f_1, \ldots, f_m)$. On the other hand, let $(a_1, \ldots, a_n) \in \mathbf{V}(f_1, \ldots, f_m)$. Every polynomial $f \in I$ can be expressed as $f = \sum_{i=1}^{m} h_i f_i$, for some $h_1, \ldots, h_m \in \mathcal{R}$. It follows that

$$\begin{aligned} f(a_1, \ldots, a_n) &= \sum_{i=1}^{m} h_i(a_1, \ldots, a_n) f_i(a_1, \ldots, a_n) \\ &= \sum_{i=1}^{m} h_i(a_1, \ldots, a_n) \cdot 0 = 0. \end{aligned}$$

Thus, $(a_1, \ldots, a_n) \in \mathbf{V}(I)$ and $\mathbf{V}(f_1, \ldots, f_m) \subseteq \mathbf{V}(I)$. Therefore, we have $\mathbf{V}(I) = \mathbf{V}(f_1, \ldots, f_m)$. ■

Proposition 3.42. *Let $V \subseteq \mathbb{F}^n$ be an affine variety. We define*

$$\mathbf{I}(V) = \{f \in \mathcal{R} : f(a_1, \ldots, a_n) = 0, \forall (a_1, \ldots, a_n) \in V\}.$$

*The set $\mathbf{I}(V)$ is an ideal of $\mathcal{R}$ called the **ideal of** V.*

Remark 3.43. *Let $f_1, \ldots, f_m \in \mathcal{R}$ and let $\langle f_1, \ldots, f_m \rangle$ be an ideal of $\mathcal{R}$. Since $f_1, \ldots, f_m$ vanish on $\mathbf{V}(f_1, \ldots, f_m)$ so does $f = \sum_{i=1}^{m} h_i f_i \in I$ where $h_i \in \mathcal{R}$. This shows that $f \in \mathbf{I}(\mathbf{V}(f_1, \ldots f_m))$ and implies that*

$$\langle f_1, \ldots, f_m \rangle \subset \mathbf{I}(\mathbf{V}(f_1, \ldots f_m)).$$

An important remark following Proposition 3.41 is the fact that affine varieties are determined by ideals. An implication of this together with Proposition 3.40 shows that in order to compute an affine variety defined by $f_1, \ldots, f_m$, one needs to obtain the right generating set for the ideal $\langle f_1, \ldots, f_m \rangle$ that gives a better description for the set of solutions. We will see that Gröbner bases with respect to lexicographic monomial ordering is the desired generating set that helps in solving a system of polynomial equations.

Definition 3.44 (ℓ-th Elimination Ideal)**.** *Let I be an ideal in $\mathbb{F}[x_1, \ldots, x_n]$. For $\ell = 0, \ldots, n-1$, the ℓ-**th elimination ideal** I_ℓ is the ideal of $\mathbb{F}[x_{\ell+1}, \ldots, x_n]$ defined by*

$$I_\ell = I \cap \mathbb{F}[x_{\ell+1}, \ldots, x_n].$$

Theorem 3.45 (The Elimination Theorem)**.** *Let $\mathbb{F}[x_1, \ldots, x_n]$ be a polynomial ring with lexicographic monomial ordering where $x_1 > x_2 > \cdots > x_n$. Let I be an ideal in $\mathbb{F}[x_1, \ldots, x_n]$ and G be a Gröbner basis of I. Then for every $\ell \in \{0, 1, \ldots, n-1\}$, the set*

$$G_\ell = G \cap \mathbb{F}[x_{\ell+1}, \ldots, x_n]$$

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

Proof. See Theorem 2 in Section §3.1 of [CLO15, pg. 122]. ■

Polynomial systems that come from cryptographic primitives are typically defined over a finite field $\mathbb{F}_q$ and have a finite number of solutions. The ideal generated by such polynomials has several algebraic properties described in the following proposition.

Proposition 3.46 (Finiteness Criterion)**.** *Let $\overline{\mathbb{F}}$ be an algebraic closure of the field $\mathbb{F}$ and let $I = \langle f_1, \ldots, f_m \rangle$ be an ideal of $\mathcal{R} = \mathbb{F}[x_1, \ldots, x_n]$. The following conditions are equivalent.*

1. *The system of equations $\{f_1, \ldots, f_m\}$ has only finitely many solutions in $\overline{\mathbb{F}}$.*
2. *For $i = 1, \ldots, n$, we have $I \cap \mathbb{F}[x_i] \neq \{0\}$.*
3. *The $\mathbb{F}$-vector space $\mathbb{F}[x_1, \ldots, x_n]/I$ is finite-dimensional.*
4. *The set $\mathsf{M}(\mathcal{R}) \setminus \mathsf{LM}(I)$ is finite.*
5. *For every $i \in \{1, \ldots, n\}$, there exists a nonnegative integer α_i such that $x_i^{\alpha_i} \in \mathsf{LT}(I)$.*

*Any ideal of $\mathcal{R}$ that satisfies one of the conditions above is called a **zero-dimensional** ideal.*

Proof. See Proposition 3.7.1 in Section §3.7 of [KR00, pg.243]. ■

Theorem 3.47 (Hilbert Nullstellensatz [CLO15])**.** *Let $\overline{\mathbb{F}}$ be an algebraically closed field. If $f, f_1, \ldots, f_m \in \overline{\mathbb{F}}[x_1, \ldots, x_n]$ then $f \in \mathbf{I}(\mathbf{V}(f_1, \ldots, f_m))$ if and only if*

$$f^s \in \langle f_1, \ldots, f_m \rangle$$

for some positive integer s.

Proof. See the proof of Theorem 2 in [CLO15, pg. 179]. ■

Definition 3.48. *Let $I \subseteq \mathcal{R}$ be an ideal of $\mathcal{R}$. The radical of I, denoted by $\sqrt{I}$, is defined to be the following set*

$$\sqrt{I} = \{f : f^s \in I \text{ for some integer } s \geq 1\}.$$

Proposition 3.49. *Let $V \subseteq \mathbb{F}^n$ be a variety and let $f \in \mathcal{R}$. If $f^s \in \mathbf{I}(V)$ for some integer $s \in \mathbb{Z}_{\geq 1}$ then $f \in \mathbf{I}(V)$.*

Proof. Let $v \in V$. If $f^s \in \mathbf{I}(V)$ then $(f(v))^s = 0$. Since $f(v) \in \mathbb{F}$, the relation $(f(v))^s = 0$ is possible only if $f(v) = 0$. Since $v \in V$ is arbitrary, then $f \in \mathbf{I}(V)$. ■

The above proposition implies that an ideal that consists of all polynomials of $\mathcal{R}$ that vanish on a variety has a property that if some power of a polynomial in $\mathcal{R}$ belongs to the ideal, then the polynomial itself must belong to the ideal. This property motivates the following definition.

Definition 3.50 (Radical Ideal)**.** *An ideal I is **radical** if $f^s \in I$ for some positive integer s implies that $f \in I$. Equivalently, $I = \sqrt{I}$.*

Let $F = \{f_1, \ldots, f_m\} \subseteq \overline{\mathbb{F}}_q[x_1, \ldots, x_n]$ where $\overline{\mathbb{F}}_q$ is the algebraic closure of $\mathbb{F}_q$. Note that, in the context of cryptanalysis, we are not interested with solutions that exist in $\overline{\mathbb{F}}_q \setminus \mathbb{F}_q$. In order to remove these unnecessary solutions, the following set is appended to F

$$\{x_i^q - x_i : \forall i = 1, \ldots, n\}.$$

This also ensures that the ideal $\langle F \cup \{x_i^q - x_i : 1 \leq i \leq n\}\rangle$ is zero-dimensional by condition 5 of Proposition 3.46. In addition to being a zero-dimensional ideal, the following Proposition shows that $\langle F \cup \{x_i^q - x_i : 1 \leq i \leq n\}\rangle$ is a radical ideal.

Proposition 3.51 (Seidenberg's Lemma)**.** *Let I be a zero-dimensional ideal in $\mathbb{F}[x_1, \ldots, x_n]$. Suppose that for every $i \in \{1, \ldots, n\}$, there exists a nonzero univariate polynomial $g_i \in I \cap \mathbb{F}[x_i]$ such that the greatest common divisor of g_i and its formal derivative is equal to 1. Then I is a radical ideal.*

Proof. See Proposition 3.7.15 in Section §3.7 of [KR00, pg.250]. ■

Definition 3.52 (Perfect Field[KR00])**.** *A field $\mathbb{F}$ is called a **perfect field** if either*

- *its characteristic is equal to 0 or*
- *its characteristic is $p > 0$ and for every $\alpha \in \mathbb{F}$ there exists $\beta \in \mathbb{F}$ such that $\beta^p = \alpha$.*

Remark 3.53. *Every finite field $\mathbb{F}_q$, where $q = p^e$ with p a prime number and $e \in \mathbb{Z}_{>0}$, is a perfect field. By definition, the characteristic of $\mathbb{F}_q$ is equal to $p > 1$. Note that the map $x \mapsto x^{p^{e-1}}$ implies that $\left(x^{p^{e-1}}\right)^p = x^q = x$ for all $x \in \mathbb{F}_q$.*

The most important implication of the ideal $\langle F \cup \{x_i^q - x_i : 1 \leq i \leq n\}\rangle$ being zero-dimensional and radical is related to the shape of its reduced Gröbner basis w.r.t. lexicographic monomial ordering. This is formally described in the following Theorem (also known as the Shape Lemma).

Theorem 3.54 (The Shape Lemma)**.** *Let $\mathbb{F}$ be a perfect field and let $\mathbb{F}[x_1, \ldots, x_n]$ be a polynomial ring with lexicographic monomial ordering. Let $I \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be a zero-dimensional radical ideal such that for any two distinct elements in $\mathbf{V}(I)$ their n-th coordinates are distinct. Also let $g_n \in \mathbb{F}[x_n]$ be the monic generator of the $(n-1)$-th elimination ideal, i.e. $\langle g_n\rangle = I_{n-1} = I \cap \mathbb{F}[x_n]$.*

1. *The reduced Gröbner basis of I is of the form*

$$\{x_1 - g_1, \ldots, x_{n-1} - g_{n-1}, g_n\}$$

where $g_1, \ldots, g_{n-1} \in \mathbb{F}[x_n]$.

2. *The polynomial g_n has $d = \deg(g_n)$ distinct zeros $a_1, \ldots, a_d \in \overline{\mathbb{F}}$ in the algebraic closure $\overline{\mathbb{F}}$ of $\mathbb{F}$. The set of zeros of I is*

$$\{(g_1(a_i), \ldots, g_{n-1}(a_i), a_i) : i = 1, \ldots, d\}.$$

Proof. See Theorem 3.7.25 in Section §3.7 of [KR00, pg.257]. ■

Following the Shape Lemma, if the reduced Gröbner basis G of an ideal I has a triangular form, then g_n is a univariate polynomial in $G \cap I_{n-1} \subseteq \mathbb{F}[x_n]$. The roots of g_n can be obtained by polynomial factorization. Once the possible solutions for x_n are found, we can substitute each value in the polynomials $x_i - g_i \in \mathbb{F}[x_i, x_n]$, for $i = 1, \ldots, n-1$, to obtain all solutions of the complete system of equations.

One particular case that is often encountered in algebraic cryptanalysis is that there exists an unique solution to the system of equations which lies in the base field. This has the following consequences on the corresponding ideal and its reduced Gröbner basis.

Proposition 3.55. *Let $F = \{f_1, \ldots, f_m\} \subset \mathcal{R} = \mathbb{F}_q[x_1, \ldots, x_n]$ and let $\tilde{I}$ be the ideal $\tilde{I} = \langle f_1, \ldots, f_m, x_1^q - x_1, \ldots, x_n^q - x_n \rangle \subset \mathcal{R}$. A solution $(a_1, \ldots, a_n) \in \mathbb{F}_q^n$ of F is the unique solution in $\mathbb{F}_q^n$ if and only if $\tilde{I} = \langle x_1 - a_1, \ldots, x_n - a_n \rangle$.*

Proof. If $(a_1, \ldots, a_n)$ is the unique solution of F in $\mathbb{F}_q^n$, then $\tilde{I} \subset \langle x_1 - a_1, \ldots, x_n - a_n \rangle$. Since the only roots of polynomials $x_i^q - x_i$ are elements of $\mathbb{F}_q$, the vector $(a_1, \ldots, a_n)$ is also the unique solution in the algebraic closure $\overline{\mathbb{F}}_q$ of $\mathbb{F}_q$ for the system of equations $F \cup \{x_i^q - x_i : i = 1, \ldots, n\}$. Theorem 3.47 states that for all $i = 1, \ldots, n$ there exists a positive integer m_i such that $(x_i - a_i)^{m_i} \in \tilde{I}$. We have $x_i - a_i = \gcd(x_i^q - x_i, (x_i - a_i)^{m_i}) \in \tilde{I}$ and therefore $\tilde{I} = \langle x_1 - a_1, \ldots, x_n - a_n \rangle$. The converse is obvious. ■

Corollary 3.56. *The reduced Gröbner basis G of $\tilde{I}$ with respect to degree-reverse lexicographic ordering is $G = \{x_1 - a_1, \ldots, x_n - a_n\}$.*

Given a Gröbner basis G of an ideal I w.r.t. a particular monomial ordering, there are algorithms that convert G to a Gröbner basis w.r.t. another monomial ordering. Two of the examples are the FGLM algorithm [FGLM93] and the Gröbner walk algorithm [CKM97]. Moreover, for a zero-dimensional ideal, the FGLM algorithm has a polynomial-time complexity in the number of solutions for the system of equations. A common strategy to solve a system of polynomial equations that corresponds to a zero-dimensional ideal I is then done in the following way: one starts by computing a Gröbner basis G of I using a monomial ordering that is considered to be efficient for computational purpose (e.g., degree-reverse lexicographic), then G is converted to another Gröbner basis G' w.r.t. lexicographic ordering using the FGLM algorithm. This, in practice, is considered to be a more efficient technique rather than computing G' directly using lexicographic ordering.

4 Gröbner Bases Algorithms

Contents

The previous chapter described Gröbner basis, its properties and some of its applications. We will now explain how to compute a Gröbner basis of an ideal $I = \langle f_1, \ldots, f_m \rangle \subseteq \mathcal{R}$ from its initial basis $\{f_1, \ldots, f_m\}$.

This chapter serves as a survey on existing Gröbner bases algorithms. We will explain three different algorithms. The first Gröbner basis algorithm is the Buchberger algorithm. It was introduced by Bruno Buchberger in his PhD thesis [Buc65, Buc06]. We give its description in Section 4.1. Since then, various attempts to improve its performance have been proposed. One of the main directions in improving the Buchberger algorithm is by detecting useless computations in advance. Some of the strategies to avoid useless computations in Gröbner bases algorithm will be mentioned in the same section.

The description of the Buchberger algorithm will be followed by the F_4 algorithm [Fau99], which is considered as a state-of-the-art Gröbner basis algorithm. One of its novel features is its ability to efficiently perform many polynomial reductions together. This is accomplished by computing the (reduced-)row echelon form of the coefficient matrix corresponding to a set of polynomial equations. We will describe the algorithm in Section 4.2.

The last section shall describe the XL Algorithm [CKPS00]. Although it was originally proposed as a way to solve overdefined system of equations, it turns out that the XL Algorithm can be seen as a redundant variant of the F_4 algorithm [AFI+04]. In that respect, the XL algorithm falls into the category of Gröbner basis algorithms. The cryptographic relevance of XL encourages us to mention it in this thesis.

This chapter, however, is not intended to be a complete survey that covers all existing Gröbner bases algorithms in the literature. One particular family of algorithms that we do not discuss is *signature-based Gröbner bases algorithms*. The concept of *signature* for a Gröbner basis algorithm was formally introduced by Faugère in the F_5 algorithm [Fau02]. Since then, there are numerous proposals of signature-based algorithms [AP10, GGV10, GIW16, EP10, EP11, RS12]. We do not cover such family of algorithms because the subject of signatures in the Gröbner bases algorithms deserves dedicated attention. Interested readers are referred to [EF17] for a recent survey on signature-based Gröbner bases algorithms.

4.1 Buchberger Algorithm

Let $I = \langle f_1, \ldots, f_m \rangle \subseteq \mathcal{R}$ be an ideal generated by $F = \{f_1, \ldots, f_m\}$. Recall that the set $G = \{g_1, \ldots, g_t\} \subseteq I$ is a Gröbner basis of I if for all $f \in I$, there exists a polynomial $g_i \in G$ such that $\mathsf{LT}(g_i) \mid \mathsf{LT}(f)$. As an element of I, each g_i can be written as a polynomial combination of $f_1, \ldots, f_m$. Thus, Gröbner bases algorithms can be viewed as a systematic approach of finding G using appropriate polynomial combinations of elements in the basis of an ideal.

The main condition of F not being a Gröbner basis of I is the existence of a polynomial $h \in I$ such that $\mathsf{LT}(f_i) \nmid \mathsf{LT}(h)$ for any $i \in \{1, \ldots, m\}$. The simplest intuition to obtain candidates for such h is by taking two distinct polynomials in the basis, say f_i, f_j with $i \neq j$, and forming a suitable combination

$$ax^\alpha f_i - bx^\beta f_j \in I, \quad a, b \in \mathbb{F},\ \alpha, \beta \in \mathbb{Z}^n_{\geq 0}$$

such that a cancellation of the leading terms occurs, leaving only smaller terms. Following this approach, Buchberger introduced the notion of S-polynomial.

Definition 4.1. *Let $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$ where $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$. Let $\gamma = (\gamma_1, \ldots, \gamma_n) \in \mathbb{Z}^n_{\geq 0}$ such that $\gamma_i = \max(\alpha_i, \beta_i)$. We say that the monomial x^γ is the **least common multiple** of x^α and x^β, denoted as $x^\gamma = \mathsf{LCM}(x^\alpha, x^\beta)$.*

Definition 4.2 (S-polynomials)**.** *Let $f, g \in \mathcal{R}$ be nonzero polynomials and let $u = \mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g))$. The **S-polynomial** of f and g is defined as*

$$\mathsf{Spoly}(f, g) = \frac{u}{\mathsf{LT}(f)} \cdot f - \frac{u}{\mathsf{LT}(g)} \cdot g.$$

Definition 4.3 (Critical Pair)**.** *For an S-polynomial $\mathsf{Spoly}(f, g)$, we call the pair $p = (f, g)$ a **critical pair**. The degree of a critical pair p is defined as $\deg(p) = \deg(\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g)))$.*

Remark 4.4. *Note that the S-polynomial $\mathsf{Spoly}(f, g)$ is defined in order to yield the desired cancellation of leading terms, i.e.*

$$\begin{aligned}
\mathsf{Spoly}(f, g) &= \frac{x^\gamma}{\mathsf{LT}(f)} \cdot f - \frac{x^\gamma}{\mathsf{LT}(g)} \cdot g \\
&= \frac{x^\gamma}{\mathsf{LT}(f)}(\mathsf{LT}(f) + \mathsf{Tail}(f)) - \frac{x^\gamma}{\mathsf{LT}(g)}(\mathsf{LT}(g) + \mathsf{Tail}(g)) \\
&= \frac{x^\gamma}{\mathsf{LT}(f)}\mathsf{Tail}(f) - \frac{x^\gamma}{\mathsf{LT}(g)}\mathsf{Tail}(g).
\end{aligned}$$

The following proposition shows that every cancellation of leading terms among polynomials with identical leading monomial is attributed to the cancellation that occur for S-polynomials.

Proposition 4.5. *Let $f_1, \ldots, f_m \in \mathcal{R}$ and $\mathsf{LM}(f_i) = x^\alpha$ for all $i = 1, \ldots, m$. Suppose we have a polynomial $h = \sum_{i=1}^m f_i$. If $\mathsf{LM}(h) < x^\alpha$, then h is a linear combination, with coefficients in $\mathbb{F}$, of the S-polynomials $\mathsf{Spoly}(f_j, f_k)$ for $1 \leq j, k \leq m$. Furthermore, we have $\mathsf{LM}(\mathsf{Spoly}(f_j, f_k)) < x^\alpha$ for each j, k.*

Proof. See Lemma 5 in Section §2.6 of [CLO15, pg. 85] ■

The notion of S-polynomials together with multivariate division and the proposition above lead to an important and easily verifiable characterization of Gröbner bases. This characterization is also a foundation for the formulation of the Buchberger algorithm.

Theorem 4.6 (Buchberger's Criterion)**.** *Let I be an ideal of the polynomial ring $\mathcal{R}$. A basis $G = \{g_1, \ldots, g_t\}$ of I is a Gröbner basis of I if and only if for all pairs $g_i \neq g_j$, the remainder after division of $\mathsf{Spoly}(g_i, g_j)$ by G (listed in some order) is zero.*

Proof. See Theorem 6 in Section §2.6 of [CLO15, pg. 86]. ■

Following Buchberger's Criterion, the question now is what can be done with the nonzero remainders resulting from S-polynomials divided by a (non-Gröbner) basis of an ideal. Recall that a remainder from multivariate division algorithm consists of terms, including the leading term, that are not divisible by any leading term of polynomials in the divisor. If I is a polynomial ideal and $g_i, g_j \in G \subseteq I$, the remainder after division of $\mathsf{Spoly}(g_i, g_j)$ by G is also an element of I. These remainders are then considered as polynomials that provide more informations about the ideal I. Thus, expanding the existing basis of an ideal by adding these remainders is the most obvious method to reach a new basis that eventually satisfies Buchberger's criterion.

Theorem 4.7 (Buchberger's Algorithm)**.** *Let $I = \langle f_1, \ldots, f_m \rangle \neq \{0\}$ be an ideal of $\mathcal{R}$. Then a Gröbner basis for I can be constructed in a finite number of steps by Algorithm 4.1.*

Proof. We first show that $G \subseteq I$ holds in every iteration. This is true in the initial step. The cardinality of G is increased by adding the remainder after division of $\mathsf{Spoly}(h_1, h_2)$ by G' with $h_1, h_2 \in G' \subseteq G$. We have $G \subseteq I$, so $h_1, h_2 \in I$. This implies that $\mathsf{Spoly}(h_1, h_2) \in I$, and since we are dividing by G', the remainder r is also in I. Hence $G \cup \{r\} \subseteq I$.

The algorithm stops when $G = G'$, that is for all $h_1, h_2 \in G$, the remainder after division of $\mathsf{Spoly}(h_1, h_2)$ by G' is equal to 0. By Theorem 4.6, the set G is a Gröbner basis of I.

For a proof of its termination, we refer to the proof of Theorem 2 in Section §2.7 of [CLO15, pg. 91]. ■

Remark 4.8. *We refer to the set G in Algorithm 4.1 as the intermediate basis of $I = \langle f_1, \ldots, f_m \rangle$.*

4.1.1 Improvements with Gebauer-Möller Installation

The Buchberger's criterion and the Buchberger's algorithm lay the foundation for the development of other Gröbner bases algorithms. We will now discuss some improvements in line with the strategy to avoid unnecessary reductions.

By "unnecessary reductions", we mean reductions that yield the zero polynomial, since these do not provide useful new information about the ideal. The approach to avoid reductions to zero occurring in the main loop of Buchberger's algorithm is by detecting them in advance. Some criteria that identify some zero-reductions beforehand are discussed below.

Input: $F = (f_1, \ldots, f_m) \subseteq \mathcal{R}$
Result: A Gröbner basis G for $I = \langle f_1, \ldots, f_m \rangle$

```
1 G ← F
2 repeat
3 |  G' ← G
4 |  forall (h1, h2) ∈ G' × G' and h1 ≠ h2 do
5 |  |  r ← FullReduce(Spoly(h1, h2), G')
6 |  |  if r ≠ 0 then
7 |  |  |  G ← G ∪ {r}
8 until G = G';
9 return G
```

Algorithm 4.1: Buchberger's Algorithm

Definition 4.9. *Let $G = \{g_1, \ldots, g_t\}$ be a finite subset of $\mathcal{R}$. We say that a polynomial $f \in \mathcal{R}$ reduces to r modulo G, written $f \underset{G}{\longrightarrow} r$, if f can be written in the form*

$$f = q_1 g_1 + \ldots + q_t g_t + r \qquad q_i, r \in R \tag{4.1}$$

such that whenever $q_i g_i \neq 0$, we have

$$\mathsf{LM}(q_i g_i) \leq \mathsf{LM}(f)$$

and r is either 0 or r is not reducible by G.

Remark. *Definition 4.9 is defined independently from the existence of an algorithm, such as Algorithm 3.1, that calculates $q_1, \ldots, q_t, r$ given f and G. That is, if r is the remainder after the division of f by G, then $f \underset{G}{\longrightarrow} r$. However, the converse is not true in general as the remainder after a division does not have to be unique.*

Definition 4.9 allows us to state a more general version of Buchberger's criterion below.

Theorem 4.10. *A basis G for an ideal I is a Gröbner basis of I if and only if $\mathsf{Spoly}(g_i, g_j) \underset{G}{\longrightarrow} 0$ for all $i \neq j$.*

Theorem 4.11 (Buchberger's First Criterion)**.** *Let G be a finite subset of the ring $\mathcal{R}$. Let $f, g \in G$ with $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g)) = \mathsf{LM}(f) \cdot \mathsf{LM}(g)$. Then*

$$\mathsf{Spoly}(f, g) \underset{G}{\longrightarrow} 0.$$

Proof. See Proposition 4 in Section §2.9 of [CLO15, pg. 106]. ■

Definition 4.12 (Standard Representation)**.** *Let $f \in \mathcal{R}$ and $G = \{g_1, \ldots, g_k\}$ be a finite subset of $\mathcal{R}$. A representation*

$$f = \sum_{i=1}^{k} \mathsf{t}_i \cdot g_i$$

with terms $\mathsf{t}_i \in \mathsf{T}(\mathcal{R})$, $g_i \in G$ pairwise different is called a standard representation if $\max\{\mathsf{LM}(\mathsf{t}_i g_i) : 1 \leq i \leq k\} \leq \mathsf{LM}(f)$.

Theorem 4.13 (Buchberger's Second Criterion)**.** *Let* $G = \{g_1, \ldots, g_k\}$ *be a finite subset of* $\mathcal{R}$ *and* $f, h_1, h_2 \in \mathcal{R}$*. If*

1. $\mathsf{LM}(f) \mid \mathsf{LCM}(\mathsf{LM}(h_1), \mathsf{LM}(h_2))$ *and*
2. $\mathsf{Spoly}(f, h_1)$ *and* $\mathsf{Spoly}(f, h_2)$ *have a standard representation w.r.t.* G*.*

then, the S-polynomial $\mathsf{Spoly}(h_1, h_2)$ *has a standard representation w.r.t.* G*.*

Proof. See Proposition 5.70 in Section §5.5 of [BW93, pg. 223]. ■

There are at least two ways to incorporate both the Buchberger's first and the second criterion inside the Buchberger's algorithm. The first one is done by tracking which critical pairs have already been selected from the list during an execution of the main loop. Following the first criterion of Buchberger, whenever two polynomials g_1, g_2 are found in the intermediate basis G such that $\mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2)) = \mathsf{LM}(g_1)\mathsf{LM}(g_2)$ the algorithm marks (g_1, g_2) as having been treated and such pair is not added into the set of critical pairs. Now suppose that a critical pair (g_1, g_2) is selected. The algorithm tries to find a polynomial $g \in G$ such that $\mathsf{LM}(g) \mid \mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2))$ and then it marks two pairs $(g, g_1), (g, g_2)$ as having been treated. If such event occurred, then nothing is done about (g_1, g_2). Otherwise, the pair (g_1, g_2) is treated similarly as in the Buchberger's algorithm (see Algorithm 4.1). The algorithm GRÖBNERNEW1 in [BW93, pg. 226] gives a more detail description of the above variant of Buchberger's algorithm.

```
   Input: Polynomials f_1, ..., f_m ∈ R
   Input: A selection function SELECT
   Result: A Gröbner basis G for the ideal ⟨f_1, ..., f_m⟩
1  for i ← 1 to m do
2      (G, P) ← UPDATE(G, P, f_i)
3  while P ≠ {} do
4      (p_1, p_2) ← SELECT(P)
5      P ← P \ {(p_1, p_2)}
6      f ← FULLREDUCE(Spoly(p_1, p_2), G)
7      if f ≠ 0 then
8          (G, P) ← UPDATE(G, P, f)
9  return G
```

Algorithm 4.2: Improved Buchberger's algorithm with Gebauer-Möller installation.

The second way of integrating Buchberger's first and second criterion is given in Algorithm 4.2. The actual implementation of both criteria is done by the UPDATE routine. Unlike the previous variant that waits until a critical pair is selected before making decision to eliminate it on the basis of the second criterion, Algorithm 4.2 tries to achieve this as early as possible. In addition to that, it also eliminates redundant elements in the intermediate basis G. Removing such redundant elements also reduces the number of critical pairs that needs to be processed.

The UPDATE function takes the intermediate basis G, the set of critical pairs P and a polynomial f as inputs. It incorporates Buchberger's first and second criterion together with elimination of redundant polynomials in G by employing four (4) **while**-loops. The first and second loops deal with the new critical pairs constructed from the input f and other elements of the intermediate basis G. The third and the fourth loop process elements in the old critical pair P and remove redundant elements in G respectively.

Definition 4.14. *Let $g_1, f, g_2 \in \mathcal{R}$ be polynomials that satisfy the equivalent conditions*

1. $\mathsf{LM}(f) \mid \mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2))$,
2. $\mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(f)) \mid \mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2))$ *and* $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g_2)) \mid \mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2))$

*then we call (g_1, f, g_2) a **Buchberger triple**.*

The first **while**-loop in the UPDATE function checks for each new critical pair (f, g_1) and tries to find another critical pair (f, g_2) on the existing list such that $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g_2)) \mid \mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g_1))$. The occurrence of such event causes elimination of the pair (f, g_1) due to (f, g_2, g_1) being a Buchberger triple. Furthermore, the pair (g_1, g_2) does not exist in this list at all. Notice that this loop keeps the pairs (f, g_1) with disjoint leading monomial. The rationale behind keeping these pairs is as follows :

> If two or more pairs in C have the same LCM of the leading monomials, so that there is a choice as to which one(s) should be deleted, then it is advantageous to try and keep one where the leading monomials are disjoint; that way, one eventually gets rid of all of them. [BW93, pg. 231]

The task of eliminating those new pairs with disjoint leading term is accomplished by the second **while**-loop, which is based on the Buchberger first criterion.

A more general version of Buchberger's criterion in Theorem 4.10 together with Buchberger second criterion tell that whenever a Buchberger triple (g_1, f, g_2) occurs in a Buchberger algorithm and both pairs $(g_1, f), (f, g_2)$ are already treated, then the critical pair (g_1, g_2) can be discarded from the computation. This is achieved by the third **while**-loop. Note that if two out of three LCM of leading monomials involved are equal, for instance

$$\mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(f)) = \mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2)),$$

then both (g_1, f, g_2) and (f, g_2, g_1) are Buchberger triples. Thus, either (g_1, g_2) or (f, g_1) can be eliminated. However, there is a risk of erroneously eliminating both pairs (the first one on account of f and the latter one on account of g_2). This loop avoids it by removing only those triples (g_1, f, g_2) such that both $\mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(f))$ and $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g_2))$ properly divide $\mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2))$.

After the updated new and old set of critical pairs are combined in P', the UPDATE function finally removes the redundant elements in G. Whenever a new element f in the ideal was found during the execution of the algorithm,

Input: An intermediate basis G
Input: A set of critical pairs P
Input: A nonzero polynomial $f \in \mathcal{R}$
Result: An updated intermediate basis G'
Result: An updated set of critical pairs P'

$C \leftarrow \{(f, g) : g \in G\}$
$D \leftarrow \{\}$
while $C \neq \{\}$ **do**
  Select (f, g_1) from C
  $C \leftarrow C \setminus \{(f, g_1)\}$
  $\mathfrak{m} \leftarrow \mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g_1))$
  if $\mathfrak{m} = \mathsf{LM}(f) \cdot \mathsf{LM}(g_1)$ **or**
  (
   $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g_2)) \not| \, \mathfrak{m} \quad \forall (f, g_2) \in C$
   and
   $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g_2)) \not| \, \mathfrak{m} \quad \forall (f, g_2) \in D$
)
   then
    $D \leftarrow D \cup \{(f, g_1)\}$
$E \leftarrow \{\}$
while $D \neq \{\}$ **do**
  Select (f, g) from D
  $D \leftarrow D \setminus \{(f, g)\}$
  if $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g)) \neq \mathsf{LM}(f)\mathsf{LM}(g)$ **then**
    $E \leftarrow E \cup \{(f, g)\}$
$P' \leftarrow \{\}$
while $P \neq \{\}$ **do**
  Select (p_1, p_2) from P
  $P \leftarrow P \setminus \{(p_1, p_2)\}$
  $\mathfrak{m} \leftarrow \mathsf{LCM}(\mathsf{LM}(p_1), \mathsf{LM}(p_2))$
  if $\mathsf{LM}(f) \not| \, \mathfrak{m}$ **or** $\mathsf{LCM}(\mathsf{LM}(p_1), \mathsf{LM}(f)) = \mathfrak{m}$ **or**
   $\mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(p_2)) = \mathfrak{m}$ **then**
    $P' \leftarrow P' \cup \{(p_1, p_2)\}$
$P' \leftarrow P' \cup E$
$G' \leftarrow \{\}$
while $G \neq \{\}$ **do**
  Select g from G
  $G \leftarrow G \setminus \{g\}$
  if $\mathsf{LM}(f) \not| \, \mathsf{LM}(g)$ **then**
    $G' \leftarrow G' \cup \{g\}$
$G' \leftarrow G' \cup \{f\}$
return G', P'

Algorithm 4.3: Gebauer-Möller Installation [BW93, GM88].

all the polynomials $g \in G$ such that $\mathsf{LM}(f) \mid \mathsf{LM}(g)$ are eliminated from G. One can justify this approach with the two arguments. First, if $\mathsf{LM}(f) \mid \mathsf{LM}(g)$ this implies that $\mathsf{LM}(f) \mid \mathsf{LCM}(\mathsf{LM}(g), \mathsf{LM}(h))$ for any $h \in \mathbb{F}[x_1, \ldots, x_n]$. Thus (g, f, h) is a Buchberger triple. The second argument is, by removing g, at the end the set G is still a Gröbner basis. In fact after the execution of the algorithm is finished, normalizing all $g \in G$ such that $\mathsf{LC}(g) = 1$ will make G a minimal Gröbner basis. However, later we will describe the drawback of this approach and we will also propose a better strategy to optimally maintain G.

The above mechanism of incorporating both Buchberger first and second criterion is due to Gebauer and Möller [GM88]. It is commonly referred as Gebauer-Möller installation. The pseudo-code is available in Algorithm 4.3.

4.2 Faugère's F_4 Algorithm

In this section another Gröbner basis algorithm called the F_4 algorithm[Fau99] is described. Generally the F_4 algorithm follows the same principles as the Buchberger algorithm, i.e., iteratively extending the existing intermediate basis by nonzero remainders after the division of S-polynomials with the current intermediate basis. Though a notable difference here is the F_4 algorithm performs the reduction of multiple S-polynomials together. This is achieved by mapping a finite set of polynomials to its corresponding coefficient matrix over $\mathbb{F}$, followed by the computation of the (reduced) row echelon form of that matrix.

Before going to the actual description of the F_4 algorithm, we first discuss the relation between polynomial reduction and linear algebra. We will define the notion of the coefficient matrix of a finite set of polynomials. This will be followed by a short theorem that shows the usefulness of row echelon reduced matrices for Gröbner bases computation. After that, we will continue by describing the basic version of the F_4 algorithm. However, this basic version generally has poor performance in practice. Thus, in the same paper Faugère also provided an enhanced version of the F_4 algorithm with better performance in practice compared to its basic version. The enhanced version is discussed in the last part of this section.

4.2.1 Linear Algebra and Gaussian Elimination of Polynomials

Let $\mathcal{M}$ be an $m \times r$ matrix defined over $\mathbb{F}$. We denote by $\mathcal{M}_{i,j} \in \mathbb{F}$ the element in row $i \in \{1, \ldots, m\}$ and column $j \in \{1, \ldots, r\}$ of $\mathcal{M}$. The notation $\mathsf{Row}(\mathcal{M}, i) \in \mathbb{F}^r$ is used to denote the i-th row of $\mathcal{M}$.

Let $M = (\mathfrak{m}_1, \ldots, \mathfrak{m}_r)$ be an ordered set of monomials in $\mathcal{R}$ and let $V_M = \{\sum_{i=1}^{r} c_i \mathfrak{m}_i : c_i \in \mathbb{F}, 1 \leq i \leq r\}$ be a vector subspace of $\mathcal{R}$ generated by M. We consider the linear map

$$\phi_M : V_M \mapsto \mathbb{F}^r \tag{4.2}$$

where $\phi_M(\mathfrak{m}_i) = e_i$ and $e_1, \ldots, e_r$ are the elements of the canonical basis of $\mathbb{F}^r$. The map ϕ_M allows an interpretation of polynomials in V_M as vectors in $\mathbb{F}^r$. On the other hand, the inverse function ϕ_M^{-1} gives an interpretation of vectors in $\mathbb{F}^r$ as polynomials in $V_M \subseteq \mathcal{R}$. We will now extend ϕ_M for a set of polynomials.

Definition 4.15. *Let $F = \{f_1, \ldots, f_m\} \subseteq \mathcal{R}$ and $M = \mathsf{M}(F)$ be the ordered set of monomials of F in descending order w.r.t. the monomial ordering $<$ in $\mathcal{R}$.*

*The **coefficient matrix** of F is the $m \times r$ matrix $\mathcal{M}$ constructed by assigning*

$$\mathsf{Row}(\mathcal{M}, i) \leftarrow \phi_M(f_i), \quad i \in \{1, \ldots, m\},$$

where ϕ_M is the linear function defined in (4.2).

Conversely, given the matrix $\mathcal{M}$, we can reconstruct the set of polynomials F below

$$F = \{\phi_M^{-1}(\mathsf{Row}(\mathcal{M}, i)) : 1 \leq i \leq r\} \setminus \{0\}.$$

The intuition behind the coefficient matrix $\mathcal{M}$ of $F \subseteq \mathbb{F}[x_1, \ldots, x_n]$ is that the element $\mathcal{M}_{i,j}$ at row i and column j is the coefficient of the monomial $\mathfrak{m}_j$ in f_i. Naturally, when no such monomial exists in f_i, then $\mathcal{M}_{i,j}$ is equal to zero. Equivalently, the rows and columns of $\mathcal{M}$ represent the polynomials and monomials of F respectively, as illustrated below:

$$\mathcal{M} = \begin{array}{c} \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{array} \begin{array}{c} \begin{array}{cccc} \mathfrak{m}_1 & \mathfrak{m}_2 & \ldots & \mathfrak{m}_r \end{array} \\ \begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \cdots & \vdots \\ * & * & \cdots & * \end{bmatrix} \end{array}$$

This implies that row operations in $\mathcal{M}$, such as addition and scalar multiplication, are essentially polynomial operations. In practice one can take advantage of efficient implementations of matrix algorithms in order to perform computations on polynomials. The most important matrix operation used in Gröbner bases computations is the computation of (reduced) row echelon form (also known as *Gaussian elimination*).

Definition 4.16. *Let $F, |F| = m$ be a finite subset of $\mathcal{R}$. Let $\mathcal{M}$ be the coefficient matrix of F and $\widetilde{\mathcal{M}}$ be the (reduced) row echelon form of $\mathcal{M}$. We say that the set*

$$\widetilde{F} = \{\phi_M^{-1}(\mathsf{Row}(\widetilde{\mathcal{M}}, i)) : 1 \leq i \leq m\} \setminus \{0\}$$

is the (reduced) row echelon form of F where $M = \mathsf{M}(F)$ is the ordered set of monomials of F.

Definition 4.16 motivates the definition of a Gaussian elimination algorithm for a finite set of polynomials F: Algorithm 4.4. The algorithm essentially computes the (reduced) row echelon form of F by applying Gaussian elimination on the coefficient matrix corresponding to F.

An important property of row echelon matrices is described in the following theorem.

Theorem 4.17. *[Fau99, pg. 65] Let F be a finite subset of $\mathbb{F}[x_1, \ldots, x_n]$ and let $\widetilde{F}$ be the (reduced) row echelon form of F. Then F^+ is defined as the following set:*

$$\widetilde{F}^+ = \{g \in \widetilde{F} : \mathsf{LM}(g) \notin \mathsf{LM}(F)\}.$$

For any subset $H \subseteq F$ such that $|H| = |\mathsf{LM}(F)|$ and $\mathsf{LM}(H) = \mathsf{LM}(F)$, the set $G = \widetilde{F}^+ \cup H$ is a triangular basis of the $\mathbb{F}$-vector space V generated by F. That is, for all $f \in V$, there exist $c_i \in \mathbb{F}$ and $g_i \in G$ such that $f = \sum_i c_i g_i$ with $\mathsf{LM}(g_1) = \mathsf{LM}(f)$ and $\mathsf{LM}(g_i) > \mathsf{LM}(g_{i+1})$.

Input: $F = \{f_1, \ldots, f_{|F|}\}$ - a finite subset of $\mathcal{R}$
Input: $\widetilde{F}$ - a (reduced) row echelon form of F
$M \leftarrow \mathsf{M}(F)$; // ordered in descending order
$\mathcal{M} \leftarrow |F| \times |M|$ zero matrix
for $i = 1$ **to** $|F|$ **do**
  $\mathsf{Row}(\mathcal{M}, i) \leftarrow \phi_M(f_i)$
$\widetilde{\mathcal{M}} \leftarrow$ the (reduced) row echelon form of $\mathcal{M}$
$\widetilde{F} \leftarrow \{\phi_M^{-1}(\mathsf{Row}(\widetilde{\mathcal{M}}, i)) : 1 \leq i \leq |F|\} \setminus \{0\}$
return $\widetilde{F}$

Algorithm 4.4: GaussianElimination algorithm.

4.2.2 Basic F_4 Algorithm

Before giving the actual description of the F_4 algorithm, we first need the following definition.

Definition 4.18. *For a given critical pair $p = (f, g) \in \mathcal{R}^2$, we define the following functions*

$$\textsc{Left}(p) = (x^\gamma / \mathsf{LM}(f), f), \tag{4.3}$$
$$\textsc{Right}(p) = (x^\gamma / \mathsf{LM}(g), g) \tag{4.4}$$

where $x^\gamma = \mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g))$. This definition is extended for a set of critical pairs $P \subset \mathcal{R}^2$, that is:

$$\textsc{Left}(P) = \bigcup_{p \in P} \{\textsc{Left}(p)\},$$
$$\textsc{Right}(P) = \bigcup_{p \in P} \{\textsc{Right}(p)\}.$$

The equation (4.3) and (4.4) in Definition 4.18 above are seen as the "left" part and the "right" part of S-polynomial $\mathsf{Spoly}(f, g)$. Note that the inversion of the leading coefficients is omitted, since this will be taken care by the computation of the row echelon form of $\textsc{Left}(P) \cup \textsc{Right}(P)$.

Similar with Buchberger algorithm, the F_4 algorithm takes as input a finite set of polynomials in $\mathcal{R}$ together with a selection function Select. Unlike in the Buchberger algorithm where the selection function only chooses one pair at a time, the feature of the F_4 algorithm to simultaneously reduce many S-polynomials at once requires the selection function to choose a subset of critical pairs. The description of the F_4 algorithm does not strictly give a concrete implementation of selection function. However, it is recommended to use normal selection strategy.

Definition 4.19. *Let $P \subset \mathcal{R}^2$ be a set of critical pairs and let $d_{\min} = \min\{\deg(p) : p \in P\}$ be the minimal degree of critical pairs in P. The **normal selection strategy** chooses a subset $P' \subseteq P$ such that*

$$P' = \{p \in P : \deg(p) = d_{\min}\}.$$

We now have the necessary information to describe the basic version of the F_4 algorithm. The pseudo-code of the basic F_4 algorithm is given in Algorithm 4.5. During the initialization step, the only notable difference compared to the Buchberger algorithm is that the F_4 algorithm introduces a loop counter d by which intermediate sets of the algorithm are indexed. The advantage of this counter d will be obvious in the improved version of the algorithm later. We will now describe the steps in the main iteration of the F_4 algorithm.

The first step in the main iteration of the F_4 algorithm is the selection of critical pairs. This is relatively straightforward given the selection function Select as input. This will be followed by the removal of selected pairs from the set of critical pairs P. Then the algorithm calls the functions Left as well as Right on the selected critical pairs and joins the two resulting sets into L_d. The elements of the set L_d are unevaluated products of the components of S-polynomials. Although this does not seem to provide an obvious advantage in the basic F_4 algorithm, the strategy that improves the performance of the F_4 algorithm later relies heavily on having these unevaluated products.

Input: A finite subset $F = \{f_1, \ldots, f_m\} \subseteq \mathcal{R}$
Input: A selection function Select
Output: A Gröbner basis of the ideal $\langle f_1, \ldots, f_m \rangle$

```
1  G ← F, F̃_0^+ ← F
2  d ← 0
3  P ← {(f_i, f_j) : 1 ≤ i < j ≤ m}
4  while P ≠ {} do
5      d ← d + 1
6      P_d ← Select(P)
7      P ← P \ P_d
8      L_d ← Left(P_d) ∪ Right(P_d)
9      F_d ← SymbolicPreprocessingBasic(L_d, G)
10     F̃_d ← GaussianElimination(F_d)
11     F̃_d^+ ← {f ∈ F̃_d : LM(f) ∉ LM(F_d)}
12     for h ∈ F̃_d^+ do
13         P ← P ∪ {{h, g} : g ∈ G}
14         G ← G ∪ {h}
15 return G
```

Algorithm 4.5: Basic version of F_4 Algorithm.

The next step in the main iteration of the F_4 algorithm is the reduction of S-polynomials in L_d. This is achieved by computing the row echelon form of a suitable set of polynomials, c.f. SymbolicPreprocessingBasic and GaussianElimination in line 9 and 10 respectively. The description of the sub-routine SymbolicPreprocessingBasic is given in Algorithm 4.6.

SymbolicPreprocessingBasic (cf. Algorithm 4.6) starts by evaluating the products $(\mathfrak{m} \cdot f)$ for all $(\mathfrak{m}, f) \in L$, which were previously constructed by calling the function Left and Right on the selected critical pairs. The goal of the algorithm is to add missing "reduction" polynomials $u \cdot g$ to F with $u \in \mathsf{M}(\mathcal{R}), g \in G$ and $\mathsf{LM}(u \cdot g) \in \mathsf{M}(F) \setminus \mathsf{LM}(F)$, until no such polynomials can be added anymore.

Input: A finite subset L of $\mathsf{M}(\mathcal{R}) \times \mathcal{R}$
Input: A finite subset G of $\mathcal{R}$
Output: A finite subset F of $\mathcal{R}$
1 $F \leftarrow \{\mathfrak{m} \cdot f : (\mathfrak{m}, f) \in L\}$
2 $D \leftarrow \mathsf{LM}(F)$
3 **while** $\mathsf{M}(F) \neq D$ **do**
4 Choose an $\mathfrak{m} \in \mathsf{M}(F) \setminus D$
5 $D \leftarrow D \cup \{\mathfrak{m}\}$
6 **if** $\exists g \in G : \mathsf{LM}(g) \mid \mathfrak{m}$ **then**
7 $F \leftarrow F \cup \{\mathfrak{m}/\mathsf{LM}(g) \cdot g\}$
8 **return** F

Algorithm 4.6: Algorithm SymbolicPreprocessingBasic.

The set D plays the role of tracking which monomials in $\mathsf{M}(F)$ have already been checked in the main loop of SymbolicPreprocessingBasic. It is initially set to $\mathsf{LM}(F)$, since initially for each $\mathfrak{m} \in \mathsf{LM}(F)$ there are already two distinct $f, f' \in F$ with $\mathsf{LM}(f) = \mathsf{LM}(f') = \mathfrak{m}$.

Lemma 4.20. *[Fau99, pg. 67] Let G be a finite subset of $\mathcal{R}$. Let F be the output of* SymbolicPreprocessingBasic(L_d, G) *and $\widetilde{F}$ be the reduced row echelon form of F. For all $h \in \widetilde{F}^+ = \{f \in \widetilde{F} : \mathsf{LM}(f) \notin \mathsf{LM}(F)\}$ we have $\mathsf{LM}(h) \notin \langle \mathsf{LM}(G) \rangle$.*

Proof. Assume that there exists $h \in \widetilde{F}^+$ such that $\mathsf{LM}(h) \in \langle \mathsf{LM}(G) \rangle$. This implies that there exists $g \in G$ such that $\mathsf{LM}(g) \mid \mathsf{LM}(h)$. Moreover, we have $\mathsf{LM}(h) \in \mathsf{M}(\widetilde{F}^+) \subset \mathsf{M}(\widetilde{F}) \subset \mathsf{M}(F)$ and h is lead-reducible by G. Hence $\mathsf{LM}(h)/\mathsf{LM}(g) \cdot g$ is inserted to F by SymbolicPreprocessingBasic which by definition of $\widetilde{F}^+$ contradicts the fact that $\mathsf{LM}(h) \notin \mathsf{LM}(F)$. ∎

Lemma 4.21. *Let G be a subset of $\mathcal{R}$ and let L be a finite subset of $\mathsf{M}(\mathcal{R}) \times \mathcal{R}$. We set $F =$* SymbolicPreprocessingBasic(L, G) *and $\widetilde{F}^+ = \{f \in \widetilde{F} : \mathsf{LM}(f) \notin \mathsf{LM}(F)\}$. For all polynomials f in the set $L' = \{\sum_i \mathfrak{c}_i \mathfrak{m}_i f_i : \mathfrak{c}_i \in \mathbb{F},\ (\mathfrak{m}_i, f_i) \in L\}$ we have*

$$f \xrightarrow[\widetilde{F}^+ \cup G]{} 0$$

Proof. Let $f \in L'$. By construction of F we have $L' \subseteq F$. Theorem 4.17 implies that f is an $\mathbb{F}$-linear combination of the triangular basis $\widetilde{F}^+ \cup H$ for a suitable $H \subseteq F$. Polynomials in H are either elements of L' or polynomials of the form $\mathfrak{m} \cdot g$ for $\mathfrak{m} \in \mathsf{M}(\mathcal{R})$ and $g \in G$. Thus f can be written in the form

$$f = \sum_i \mathfrak{c}_i f_i + \sum_j \mathfrak{c}_j \mathfrak{m}_j g_j, \quad \mathfrak{c}_i, \mathfrak{c}_j \in \mathbb{F}$$

and $f_i \in \widetilde{F}^+, \mathfrak{m}_j \in \mathsf{M}(\mathcal{R}), g_j \in G$. Thus, $f \xrightarrow[\widetilde{F}^+ \cup G]{} 0$. ∎

Theorem 4.22. *[Fau99, pg. 68] Given a finite subset $F \subseteq \mathcal{R}$, the F_4 algorithm returns a Gröbner basis G of an ideal $\langle F \rangle$, with $F \subseteq G$ and $\langle G \rangle = \langle F \rangle$.*

Proof. **Termination.** Assume that the **while**-loop does not terminate. There exists an ascending sequence $d_i > 0$ such that $\widetilde{F}_{d_i}^+ \neq \{\}$ for all i. Let $q_i \in \widetilde{F}_{d_i}^+$ and let $U_i = U_{i-1} + \langle \mathsf{LM}(q_i) \rangle$ with $U_0 = \{0\}$ for $i > 1$. It is trivial to show that U_i is an ideal of $\mathcal{R}$. Because polynomials in $\widetilde{F}_{d_i}^+$ are added to G at the end of every loop, by Lemma 4.20 we have $U_{i-1} \not\subseteq U_i$. This contradicts the fact that $\mathcal{R}$ is a Noetherian ring.

Correctness. The set G is constructed by $G = \bigcup_{d \geq 0} \widetilde{F}_d^+$. We claim that both statements below are loop-invariants of the **while**-loop.

1. The set $G \subset \mathcal{R}$ satisfies $F \subseteq G \subseteq \langle F \rangle$.
2. For all $g_1, g_2 \in G$ such that $(g_1, g_2) \notin P$, we have $\mathsf{Spoly}(g_1, g_2) \underset{G}{\longrightarrow} 0$.

We will now prove the first claim. Since $\widetilde{F}_0^+ = F$, we have $F \subseteq G$. Now suppose that F_d is the output of SymbolicPreprocessingBasic. Polynomials in F_d are of the form $\mathfrak{m} \cdot g$ with $\mathfrak{m}$ a monomial and $g \in G$. Since $\widetilde{F}_d$ consists of $\mathbb{F}$-linear combination of polynomials in F_d, so does $\widetilde{F}_d^+$. This shows that $G = \bigcup_{d \geq 0} \widetilde{F}_d^+ \subseteq \langle F \rangle$.

To prove the second claim, note that for $g_1, g_2 \in G$ such that $(g_1, g_2) \notin P$ this means that the critical pair (g_1, g_2) has been selected in a previous iteration, say iteration i, by the function Select. This implies that $(\mathfrak{m}_1, g_1), (\mathfrak{m}_2, g_2) \in L_i$ for some monomials $\mathfrak{m}_1, \mathfrak{m}_2$. Thus $\mathsf{Spoly}(g_1, g_2)$ is an element of the $\mathbb{F}$-vector space generated by the set $\{\mathfrak{m} \cdot f : \forall (\mathfrak{m}, f) \in L_i\}$. Hence by Lemma 4.21, we have $\mathsf{Spoly}(g_1, g_2) \underset{G}{\longrightarrow} 0$.

■

4.2.3 The Improved F_4 Algorithm

While the description of the basic F_4 algorithm computes a Gröbner basis for a given set of polynomials, the algorithm's efficiency can be significantly improved. There are two important techniques used to improve the basic F_4 algorithm.

One can check that the previous description of the F_4 algorithm keeps all critical pairs and redundant elements in the intermediate basis. The first improvement of the F_4 algorithm is the incorporation of the Buchberger's first and the second criterion. Faugère suggested the standard implementation of Gebauer-Möller installation [GM88] (see Algorithm 4.3). Instead of adding all critical pairs to P, the only pairs added are the ones that remain unaffected by the Buchberger's first and the second criterion implemented in the Update function.

The second important improvement to the F_4 algorithm is to reuse the computational results from previous iterations. In the basic version of F_4, the polynomials in $\widetilde{F}_d$ with leading monomials in $\mathsf{LM}(F_d)$ are simply discarded. These polynomials are the fully-reduced polynomials from $\tilde{F}_d$ and discarding them may lead to their recomputation in the next iterations. In the improved version of the F_4 algorithm, these polynomial are kept to replace some products $\mathfrak{m} \cdot f$ with a new product $\mathfrak{m}' \cdot f'$ with $\mathfrak{m}, \mathfrak{m}'$ be monomials of $\mathcal{R}$ such that $\mathfrak{m}' \leq \mathfrak{m}$ and $\mathsf{LM}(\mathfrak{m} \cdot f) = \mathsf{LM}(\mathfrak{m}' \cdot f')$. This means that more reductions are applied to f' already. This is achieved by the subroutine Simplify (Algorithm 4.9). The inputs to the Simplify function are a monomial $\mathfrak{m}$, a polynomial $f \in \mathcal{R}$, and a

Input: A finite subset $F = \{f_1, \ldots, f_m\} \subseteq \mathcal{R}$
Input: A selection function SELECT
Output: A Gröbner basis of the ideal $\langle f_1, \ldots, f_m \rangle$

```
1  G ← {}, P ← {}
2  d ← 0
3  while F ≠ {} do
4      Choose an f ∈ F
5      F ← F \ {f}
6      (G, P) ← UPDATE(G, P, f)
7  while P ≠ {} do
8      d ← d + 1
9      P_d ← SELECT(P)
10     P ← P \ P_d
11     L_d ← LEFT(P_d) ∪ RIGHT(P_d)
12     F_d ← SYMBOLICPREPROCESSING(L_d, G, (F_1, ..., F_{d-1}))
13     F̃_d ← GAUSSIANELIMINATION(F_d)
14     F̃_d^+ = {f ∈ F̃_d : LM(f) ∉ LM(F_d)}
15     for h ∈ F̃_d^+ do
16         (G, P) ← UPDATE(G, P, h)
17 return G
```

Algorithm 4.7: An improved F_4 algorithm.

$(d-1)$-tuple of all previously constructed F, i.e. $(F_1, \ldots, F_{d-1})$. The SIMPLIFY function is called while constructing F_d from L_d, that is during the execution of symbolic preprocessing. Some necessary changes are applied to the previously SYMBOLICPREPROCESSINGBASIC into the SYMBOLICPREPROCESSING described in Algorithm 4.8.

Lemma 4.23. *[Fau99] Let $(\mathfrak{m}, f) \in \mathsf{M}(\mathcal{R}) \times \mathcal{R}$. If $(\mathfrak{m}', f') \in \mathsf{M}(\mathcal{R}) \times \mathcal{R}$ is an output of* SIMPLIFY$(\mathfrak{m}, f, (F_1, \ldots, F_{d-1}))$*, then $\mathsf{LM}(\mathfrak{m}' \cdot f') = \mathsf{LM}(\mathfrak{m} \cdot f)$. Furthermore, if V is an $\mathbb{F}$-vector space generated by $\left(\bigcup_{i=1}^{d-1} F_i\right) \bigcup \left(\bigcup_{i=1}^{d-1} \widetilde{F_i}\right)$, then there exists a nonzero $\mathfrak{c} \in \mathbb{F}$ and $r \in V$ such that $\mathfrak{m}f = \mathfrak{c}\mathfrak{m}'f' + r$ with $\mathsf{LM}(r) < \mathsf{LM}(\mathfrak{m} \cdot f)$*

Proof. See Lemma 2.3 in [Fau99, pg. 71]. ■

Theorem 4.24. *[Fau99] Let F be a finite subset of $\mathcal{R}$ and let $\mathcal{F} = (F_1, \ldots, F_{d-1})$ with $F_i \subseteq \mathcal{R}$ for $i = 1, \ldots, d-1$. Let $(g_1, g_2) \subset \mathcal{R}^2$ be a critical pair. For $i = 1, 2$ we define $\mathfrak{m}_i = x^\gamma / \mathsf{LM}(g_i)$ with $x^\gamma = \mathsf{LCM}(\mathsf{LM}(g_1), \mathsf{LM}(g_2))$. If the following conditions hold*

1. *For $k = 1, \ldots, d-1$, we have $F_k = \{\mathfrak{m} \cdot f : \forall (\mathfrak{m}, f) \in L_k\}$ where L_k is a finite subset of $\mathsf{M}(\mathcal{R}) \times \mathcal{R}$,*
2. *For $k = 1, \ldots, d-1$, $\widetilde{F_k} \subset G$,*
3. *$f_i = \mathfrak{m}_i' \cdot g_i'$ where $(\mathfrak{m}_i', g_i') =$ SIMPLIFY$(\mathfrak{m}_i, g_i, \mathcal{F})$ for $i = 1, 2$,*
4. *$\mathsf{Spoly}(f_1, f_2) \underset{F}{\longrightarrow} 0$*

then we have $\mathsf{Spoly}(g_1, g_2) \underset{F}{\longrightarrow} 0$.

Proof. See Theorem 2.4 in [Fau99, pg. 71]. ■

Input: A finite subset L of $\mathsf{M}(\mathcal{R}) \times \mathcal{R}$
Input: A finite subset G of $\mathcal{R}$
Input: A tuple $(F_1, \ldots, F_{d-1})$ where $F_i \subseteq \mathcal{R}$
Output: A finite subset $\tilde{F}$ of $\mathcal{R}$

```
F ← {m' · f' : (m, f) ∈ L, (m', f') ← SIMPLIFY(m, f, (F_1, ..., F_{d-1}))}
D ← LM(F)
while M(F) ≠ D do
    Choose an m ∈ M(F) \ D
    D ← D ∪ {m}
    if ∃g ∈ G : LM(g) | m then
        (m', g') ← SIMPLIFY(m/LM(g), g, (F_1, ..., F_{d-1}))
        F ← F ∪ {m' · g'}
return F
```

Algorithm 4.8: Algorithm SymbolicPreprocessing

Input: A monomial $\mathfrak{m} \in \mathsf{M}(R)$
Input: A polynomial $f \in \mathcal{R}$
Input: A tuple $(F_1, \ldots, F_{d-1})$, $F_i \subseteq \mathcal{R}$, $1 \leq i \leq d-1$
Output: An element of $\mathsf{M}(\mathcal{R}) \times \mathcal{R}$

```
for u ∈ M(R) : u | m do
    if ∃j ∈ {1, ..., d − 1} : uf ∈ F_j then
        F̃_j ← row echelon form of F_j
        There exists a p ∈ F̃_j s.t. LM(p) = LM(uf)
        if u ≠ m then
            return SIMPLIFY(m/u, p, (F_1, ..., F_{d-1}))
        else
            return (1, p)
return (m, f)
```

Algorithm 4.9: Algorithm Simplify

4.3 Extended Linearization (XL) Algorithm

In this subsection we discuss the *extended linearization* (XL) algorithm to solve a system of multivariate polynomial equations. The algorithm was proposed by Courtois *et al.* [CKPS00] and it received a lot of attention in the cryptographic community due to its relatively straightforward description as well as its simplicity. The XL algorithm takes into account that the systems of equations appear in cryptanalysis of cryptographic schemes are, in general, overdefined. The authors of XL argued that these cases are not well-utilized by Gröbner bases

algorithms and they claimed that the XL algorithm is a major improvement to solve overdefined system of equations.

The theoretical foundation of the XL algorithm is build upon the linearization technique. Let us now briefly discuss the idea of linearization. Consider a random homogeneous quadratic polynomial equations F over a field $\mathbb{F}$ in variables $x_1, \ldots, x_n$ with $\binom{n}{2} = n(n-1)/2$ equations. The main idea of linearization is to transform F into a system of linear equations by treating each monomial $x_i x_j$ in F as a new independent variable, say, y_{ij}. The new linear system has $n(n-1)/2$ variables with $n(n-1)/2$ equations. This can be efficiently solved using Gaussian elimination. As soon as all solutions for y_{ij} were found, two possible values for x_i are extracted by computing the square root of y_{ii} in $\mathbb{F}$. The values y_{ij} are used to remove pathological solutions by combining the correct square roots of y_{ii} and y_{jj}.

The linearization works well when there are sufficient number of linearly independent equations. To increase the number of equations, Kipnis and Shamir introduced a new method called relinearization [KS99]. The main principle of relinearization is that variables y_{ij} in the linear equations are related rather than independent. For instance, a relation that can be derived is based on the commutativity of multiplication. Let $i, j, k, \ell \in \{1, \ldots, n\}$, we have the following relation

$$(x_i x_j)(x_k x_\ell) = (x_i x_k)(x_j x_\ell) = (x_i x_\ell)(x_j x_k).$$

This implies that $y_{ij} y_{k\ell} = y_{ik} y_{j\ell} = y_{i\ell} y_{jk}$ and the following non-linear quadratic equations can be added to the set of linear equations

$$\begin{aligned} y_{ij} y_{k\ell} - y_{ik} y_{j\ell} &= 0, \\ y_{ij} y_{k\ell} - y_{i\ell} y_{jk} &= 0, \\ y_{ik} y_{j\ell} - y_{i\ell} y_{jk} &= 0. \end{aligned}$$

Different variants of relinearization technique based on other relations, such as recursive and commutativity of multiplication of higher degree monomials, were also discussed in [KS99].

Let $F = \{f_1, \ldots, f_m\}$ be a set of quadratic polynomials in $x_1, \ldots, x_n$ over the field $\mathbb{F}$. Given a positive integer D, the XL algorithm adds polynomials of the form $\mathfrak{m} \cdot f_i$ to F, where $\mathfrak{m}$ is a monomial, such that $\deg(\mathfrak{m} \cdot f_i) \leq D$. The algorithm proceeds to solve the new equation system using the linearization technique. The complete description is given below:

1. **Multiply** : Generate all polynomials of the form $\mathfrak{m} \cdot f_i$ where $\mathfrak{m}$ is a monomial such that $\deg(\mathfrak{m} \cdot f_i) \leq D$ for all $i \in \{1, \ldots, m\}$.

2. **Linearize** : Consider an ordering on monomials such that the univariate monomials are the smallest monomials, i.e., these will be eliminated last. We treat each monomial as a new variable independent from each other.

3. **Reduction** : Compute the (reduced) row echelon form of the corresponding coefficient matrix of the linearized system of equations.

4. **Solve** : Assume the previous step yields a univariate polynomial h in the variable x_i. Find all the roots of h in $\mathbb{F}$, e.g., by polynomial factorization.

5. **Repeat** : Simplify the original system of equations by substituting x_i and repeat similar process to obtain solutions for other variables.

Based on experimental observation [CKPS00], the authors suggested the following value for D.

	D
$m = n$	2^n
$m = n + 1$	n
$m = n + 2$	$\sqrt{n} + C$ (C is a constant)

Table 4: *Suggested values for D in XL.*

The XL algorithm was intended to be an efficient algorithm to solve a system of equations F with n variables over a finite field $\mathbb{F}_q$ in which F has only one solution in $\mathbb{F}_q^n$. Note that F may have another solution in $\mathbb{F}_{q^e}^n$ where $\mathbb{F}_{q^e}$ is the e-th degree extension of $\mathbb{F}_q$. Elimination of solutions in $\mathbb{F}_{q^e}^n \setminus \mathbb{F}_q^n$ is done by inserting polynomials $\{x_i^q - x_i : i = 1, \ldots, n\}$ to F.

4.3.1 XL as a Variant of the F_4 Algorithm

In this subsection, we briefly discuss the relation between the XL algorithm and the F_4 algorithm where the former is seen as a redundant variant of the latter. The description of the XL algorithm as a variant of the F_4 relies on the following definitions.

Definition 4.25. *For a critical pair $p = (f, g) \in \mathcal{R}^2$ and a nonnegative integer $d \in \mathbb{Z}_{\geq 0}$ we define the following function*

$$\textsc{Left}_{\text{XL}}(p, d) = \{(\mathfrak{m}, f) : \mathfrak{m} \in \mathsf{M}(\mathcal{R}), \deg(\mathfrak{m} \cdot f) \leq d\},$$
$$\textsc{Right}_{\text{XL}}(p, d) = \{(\mathfrak{m}, g) : \mathfrak{m} \in \mathsf{M}(\mathcal{R}), \deg(\mathfrak{m} \cdot g) \leq d\}.$$

For a set of critical pairs $P \subseteq \mathcal{R} \times \mathcal{R}$, we also define a natural extension of above definitions as follows

$$\textsc{Left}_{\text{XL}}(P, d) = \bigcup_{p \in P} \textsc{Left}_{\text{XL}}(p, d),$$
$$\textsc{Right}_{\text{XL}}(P, d) = \bigcup_{p \in P} \textsc{Right}_{\text{XL}}(p, d).$$

Definition 4.26. *Let $P \subseteq \mathcal{R} \times \mathcal{R}$ be a set of critical pairn. For nonnegative integer $d \in \mathbb{Z}_{\geq 0}$ we define the following selection function for XL*

$$\textsc{Select}_{\text{XL}}(P, d) = \{p \in P : \deg(p) \leq d\}.$$

The description of the XL algorithm in an F_4-like fashion is given in Algorithm 4.10. Note that the algorithm described differs slightly from the original description of the XL algorithm. While the original algorithm sets the degree bound D globally at once, Algorithm 4.10 determines the optimal value for D iteratively. The **Multiply** step of the XL algorithm corresponds to the computation of L_d in line 9. The **Reduction** step of the XL algorithm corresponds to the Gaussian elimination in line 10. Algorithm 4.10 clearly shows that the XL algorithm is a redundant variant of the F_4 algorithm since the function $\textsc{Left}_{\text{XL}}$ and $\textsc{Right}_{\text{XL}}$ collect more polynomials than the function $\textsc{Left}$ and $\textsc{Right}$ of F_4. This implies that there are more polynomials constructed in F_d in the XL algorithm compared to the same set in the F_4 algorithm.

Input: A finite subset $F = \{f_1, \ldots, f_m\} \subseteq \mathcal{R}$
Output: A finite subset $G \subseteq \langle f_1, \ldots, f_m \rangle$

$G \leftarrow F$, $\tilde{F}_0^+ \leftarrow F$
$d \leftarrow 0$
$P \leftarrow \{(f_i, f_j) : 1 \leq i < j \leq m\}$
while $P \neq \{\}$ **do**
  $d \leftarrow d + 1$
  $P_d \leftarrow \textsc{Select}_{\text{XL}}(P, d)$
  $P \leftarrow P \setminus P_d$
  $L_d \leftarrow \textsc{Left}_{\text{XL}}(P_d, d) \cup \textsc{Right}_{\text{XL}}(P_d, d)$
  $F_d \leftarrow \{\mathfrak{m} \cdot f : (\mathfrak{m}, f) \in L_d\}$
  $\widetilde{F}_d \leftarrow \textsc{GaussianElimination}(F_d)$
  $\widetilde{F}_d^+ \leftarrow \{f \in \widetilde{F}_d : \mathsf{LM}(f) \notin \mathsf{LM}(F_d)\}$
  for $h \in \widetilde{F}_d^+$ **do**
    $P \leftarrow P \cup \{(h, g) : g \in G\}$
    $G \leftarrow G \cup \{h\}$
return G

Algorithm 4.10: An F_4-like description of XL algorithm.

Theorem 4.27. *Let $F \subseteq \mathcal{R}$. Then Algorithm 4.10 computes a Gröbner basis G of the ideal $\langle F \rangle \subseteq \mathcal{R}$ such that $F \subseteq G$.*

Proof. See Theorem 3 in [AFI+04]. ■

4.3.2 The Advantage of XL Algorithm

Although the XL algorithm can be viewed as a variant of the F_4 algorithm, its distinct advantage is that it produces a sparse coefficient matrix. This sparsity arises from the redundant polynomials added to the rows of the coefficient matrix as a result of the multiply step. Consequently, the XL algorithm can leverage dedicated sparse linear algebra techniques, especially Wiedemann [Wie86] and Block Wiedemann [Cop94], which are tailored for matrices over finite fields. This property makes XL particularly appealing in cryptanalytic applications involving sparse and overdefined systems of equations.

Part III
Contributions and Main Results

5 M4GB: An Efficient Gröbner Bases Algorithm

Contents

In algebraic cryptanalysis the most crucial step is solving a system of polynomial equations. Since the classical Buchberger algorithm, the development of Gröbner bases algorithms has reached significant improvements, notably with the F_4 [Fau99] and the F_5 [Fau02] algorithm.

Even though the theoretical development of Gröbner bases algorithms has achieved significant progress, many of their efficient implementations remain as closed-source binaries such as the implementation of the F_4 algorithm in FGb [Fau10] and Magma [BCP97]. Note, that the original papers of the F_4 and the F_5 algorithm did not discuss how both algorithms should be efficiently implemented. This is in contrast to the fact that their efficiency were proven by means of computer implementation.

We believe it should be the norm to release the source code of academic implementations belonging to a paper as well. As often important implementation details are not discussed in the paper, the source code is thereby necessary to fully understand and verify the results. Moreover, importantly it facilitates scientific progress and gives a thorough understanding of the problem from practical point-of-view. Two important examples of open-source community development in the domain of lattice reduction algorithms are FPLLL [dt16] and G6K [ADH+19].

On the other hand, the rise of multivariate public-key cryptography (MPKC) as one of the candidates for post-quantum cryptography requires a better understanding of the ability of Gröbner bases algorithms to solve MQ problems in practice. Thus, there is a need to revisit and understand the internal details of Gröbner bases algorithms in order to optimize it against specific types of systems of polynomial equations, in particular systems of polynomial equations underlying MPKC, i.e., dense and overdefined.

This chapter introduces a new Gröbner bases algorithm, whose name is M4GB. The M4GB algorithm is designed to consistently maintain tail-reduced polynomials, i.e., polynomials whose tail is not reducible by the intermediate

basis. This strategy is employed to avoid redundant work in the reduction step of the improved version of the F_4 algorithm. Specifically, the M4GB algorithm can be seen as operating on submatrices of the F_4's coefficient matrices, namely over only those columns which belong to non-reducible monomials. To achieve this, we also introduce a new reduction algorithm that performs an in-place reduction on polynomials of the form $\mathtt{t} \cdot f$, where $\mathtt{t}$ is a term and f is a polynomial, without explicit construction of the polynomial $\mathtt{t} \cdot f$ itself.

The M4GB algorithm is an extension of the Buchberger algorithm and it is described more natively so that one can easily derive an implementation from the high-level description of the algorithm. We have implemented a version of the M4GB algorithm optimized for dense and overdefined quadratic systems of equations over finite fields. We compare our implementation of the M4GB algorithm against three other implementations of Gröbner bases algorithm: FGb [Fau10], Magma [BCP97], and OpenF4 [CVJ]. The benchmarks are based on multivariate polynomial equations that represents multivariate public-key and signature cryptosystems. We observed that the implementation of the M4GB algorithm uses the least total CPU time and the least memory usage among these implementations, often by a significant factor. This is also proven by the ability of the M4GB algorithm to set new records in the computational challenge related to the hardness of MQ problem [YDH+15a]. The result of this chapter is a joint work with Marc Stevens presented at International Symposium on Symbolic and Algebraic Computation (ISSAC) 2017 [MS17].

Related Work. Since the proposal of the Buchberger algorithm [Buc65, Buc06], the progress on the development of Gröbner bases algorithm reached a new direction with the introduction of the F_4 algorithm [Fau99]. The efforts to improve Gröbner bases computation before the introduction of the F_4 algorithm were mainly focused around detecting useless computation in advance [GM88, Buc79]. The F_4 algorithm improves the computation of Gröbner bases by parallelizing the reduction of multiple S-polynomials at once using efficient implementations of (sparse) linear algebra. The reduction algorithm constructs a coefficient matrix corresponding to a finite set of polynomials, followed by the computation of its (reduced-)row echelon form. Moreover the existing criteria to remove unnecessary computation, such as the Gebauer-Möller installation [GM88], can be easily integrated to the F_4 algorithm. The algorithm achieved a significant performance improvement compared to the Buchberger algorithm which was demonstrated by its ability to compute a Gröbner basis for the Cyclic-9 root problem for the first time in practice. Later, Faugère introduced the concept of polynomial *signature* and proposed a new Gröbner bases algorithm called F_5 [Fau02]. The main advantage of the F_5 algorithm over the F_4 algorithm is its ability to avoid zero reductions when the input polynomials are a regular sequence.* The efficiency of the F_5 algorithm was demonstrated by its ability to compute a Gröbner basis for the Cyclic-10 problem.

The description of both the F_4 and the F_5 algorithms leaves several degrees of freedom. For instance, the selection strategy and the choice of reductor were left unexplained. This leads to the development of numerous variants of the F_4 and the F_5 algorithms. For instance, Joux and Vitse proposed a heuris-

* An ordered set $(f_1, \ldots, f_m) \subset \mathcal{R}$ is regular if f_i is not a zero divisor in the quotient ring $\mathcal{R}/\langle f_1, \ldots, f_{i-1}\rangle$ for all $i = 1, \ldots, m$. Equivalently if there exists $g \in \mathcal{R}$ such that $gf_i \in \langle f_1, \ldots, f_{i-1}\rangle$ then g belongs to $\langle f_1, \ldots, f_{i-1}\rangle$ [BFS04, Definition 1].

tic and probabilistic variant of the F_4 algorithm in [JV11] which was claimed to be suitable for algebraic attacks. Another heuristic adaptation of the F_4 algorithm for cryptanalytic purposes was also proposed in [HB13]. The landscape of variants of the F_5 algorithm is even more diverse than for the F_4 algorithm [AP10, FJ03, GGV10, GIW16, EP10, EP11, HA10]. An extensive survey on variants of the F_5 algorithm is described by Eder and Faugère in [EF17].

The F_4 algorithm is implemented in various computer algebra software, such as Maple [Map18], Magma [BCP97], Singular [DGPS18] and SageMath [The18]. Other implementations of the F_4 algorithm are standalone implementations, such as FGb [Fau10] and OpenF4 [CVJ]. The implementations dedicated for Boolean polynomials, i.e., polynomials in $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle$, are available in Magma [BCP97] and PolyBoRi [BD09a]. These implementations are particularly suitable for algebraic cryptanalysis of bit-based symmetric-key primitives such as block ciphers and stream ciphers. An implementation of the F_4 algorithm dedicated for shared and distributed memory architecture is described in [Neu13], which focuses on reducing memory usage and avoiding communication overhead. Another implementation of the F_4 algorithm described by Pierce and Monagan in [MP15] also parallelizes the construction of coefficient matrices. Finally, the computational techniques and data structures for Gröbner bases algorithms that employ polynomial signatures is explained in [RS12].

Outline Although it is presented as a variant of the Buchberger's algorithm, the strategy to maintain tail-reduced polynomials can be seen as an improvement of the Simplify function in the reduction step of the F_4 algorithm algorithm. We first revisit the reduction step of the F_4 algorithm together with the Simplify function in Section 5.1. It will be followed by the description of a reduction algorithm in Section 5.2 that prioritizes the reduction of every newly constructed reductor in a recursive manner. We generalize this reduction algorithm to perform an in-place reduction on polynomials of the form $\mathtt{t} \cdot f$, where $\mathtt{t}$ is a term and f is a polynomial. This generalization plays a role in maintaining tail-reduced polynomials and it is employed in the reduction of S-polynomials inside the M4GB algorithm, which will be described in Section 5.3. We discuss our implementation of the M4GB algorithm tailored for dense quadratic overdefined system of equations in Section 5.6. Finally, the benchmark and performance comparison of the M4GB algorithm against several existing implementations of Gröbner bases algorithm is given in Section 5.7. The discussion on the records of MQ challenges solved by the M4GB algorithm will be given in Chapter 6.

5.1 Revisiting the Reduction Step of the F_4 Algorithm

Recall that the original paper of the F_4 algorithm describes two different variants of the algorithm. The first variant is described without any criteria to remove useless critical pairs. The second variant uses two additional routines to improve its performance: Update and Simplify (see Algorithm 4.3 and Algorithm 4.9). The Update function is integrated directly in the main iteration of the F_4 algorithm, whereas the Simplify function is incorporated as a subroutine inside the SymbolicPreprocessing.

Let $I = \langle f_1, \ldots, f_m\rangle \subset \mathcal{R}$ be an ideal of $\mathcal{R}$ generated by $f_1, \ldots, f_m \in \mathcal{R}$. The goal of the Update function is to remove unnecessary critical pairs that appear during the computation of a Gröbner basis of I using the Buchberger's criteria.

Faugerè suggested to use the Gebauer-Möller installation [GM88]. On the other hand, the task of the SIMPLIFY function is to replace polynomials of the form $\mathfrak{m} \cdot f$ by a new "equivalent" and "more reduced" polynomial of the form $\mathfrak{m}' \cdot f'$ such that $\mathfrak{m}' \leq \mathfrak{m}$ where $\mathfrak{m}, \mathfrak{m}' \in \mathsf{M}(\mathcal{R})$ and $f, f' \in I$. The notion of "equivalence" here is that the polynomials $\mathfrak{m}' \cdot f'$ satisfy $\mathsf{LM}(\mathfrak{m} \cdot f) = \mathsf{LM}(\mathfrak{m}' \cdot f')$ as stated in Lemma 4.23. The paper also remarked that in 95% of cases, the monomial $\mathfrak{m}'$ is a variable in the ring $\mathcal{R}$ and $\mathfrak{m}'$ is often equal to x_n [Fau99, Remark 2.6]. At this point, however, the advantage of incorporating the SIMPLIFY function on the performance of the improved F_4 algorithm is not obvious. Our next aim is to derive an intuitive explanation on the performance gain of the F_4 algorithm when using the SIMPLIFY function.

At iteration d, recall that the SIMPLIFY function takes as input $\mathfrak{m} \in \mathsf{M}(\mathcal{R})$, $f \in I$ and a tuple $(F_1, \ldots, F_{d-1})$ such that $F_i \subset \mathcal{R}$ for all $i = 1, \ldots, d-1$. Each F_i corresponds to the constructed coefficient matrix at the i-th iteration. If $\mathfrak{m}$ has no nontrivial divisor, then it returns $(\mathfrak{m}, f)$. Otherwise, for all divisors $\mathfrak{u}$ of $\mathfrak{m}$, it checks if there exists $j \in \{1, \ldots, d-1\}$ such that $\mathfrak{u} \cdot f \in F_j$. This implies that there exists a polynomial p in the (reduced) row echelon form $\tilde{F}_j$ of F_j such that $\mathsf{LM}(p) = \mathsf{LM}(\mathfrak{u} \cdot f)$. When $\mathfrak{u}$ is a proper divisor of $\mathfrak{m}$, then it recursively calls SIMPLIFY$(\mathfrak{m}/\mathfrak{u}, p, (F_1, \ldots, F_{d-1}))$. Otherwise, the function returns $(1, p)$. Suppose that such j exists. We then have

$$\mathsf{LM}(\mathfrak{m}/\mathfrak{u} \cdot p) = \mathsf{LM}(\mathfrak{m}/\mathfrak{u} \cdot \mathfrak{u} f) = \mathsf{LM}(\mathfrak{m} \cdot f)$$

The polynomial $p \in \tilde{F}_j$ is the result of a reduction of the polynomial $\mathfrak{u} \cdot f \in F_j$ where $\mathsf{LM}(\mathfrak{u} \cdot f) = \mathsf{LM}(p)$. More precisely, it is the result of performing tail reduction on $\mathfrak{u} \cdot f$. We can now provide a more intuitive explanation of the purpose of the SIMPLIFY function in the following remark.

Remark 5.1. *Let $I = \langle f_1, \ldots, f_m \rangle \subset \mathcal{R}$ be an ideal of $\mathcal{R}$ generated by polynomials $f_1, \ldots, f_m \in \mathcal{R}$. Given a monomial $\mathfrak{m} \in \mathsf{M}(\mathcal{R})$ and a nonzero polynomial $f \in I$, the goal of* SIMPLIFY *function is to find a nonzero polynomial $g \in I$ such that $\mathsf{LM}(g) = \mathsf{LM}(\mathfrak{m} f)$ and g is a polynomial in I that can be found with the least number of terms in its tail. Thus,* SYMBOLICPREPROCESSING *returns a sparser coefficient matrix or, if possible, a smaller coefficient matrix than* SYMBOLICPREPROCESSINGBASIC. *This effectively reduces the number of field operations when computing the (reduced)-row echelon form of the coefficient matrix.*

Remark 5.2. *For Gröbner bases computations over fields, it is unnecessary to store two distinct nonzero polynomials $f, f' \in \mathcal{R}$ such that $\mathsf{LM}(f) = \mathsf{LM}(f')$ since the reducibility of a term $\mathfrak{t}$ only depends on the divisibility of the monomial of $\mathfrak{t}$ by $\mathsf{LM}(g)$ for some $g \in G$ where G is the intermediate basis.*

However, the SIMPLIFY function is implemented at the expense of storing all previously constructed coefficient matrices $F_1, \ldots, F_{d-1}$ together with their corresponding (reduced)-row echelon form $\tilde{F}_1, \ldots, \tilde{F}_{d-1}$. Moreover, the way to find an "equivalent" representation for $\mathfrak{m} \cdot f$ is done by constructing a finite sequence $(\mathfrak{m}_k, f_k)$ such that $(\mathfrak{m}_0, f_0) = (\mathfrak{m}, f)$ and $\mathfrak{m}_{k+1} \leq \mathfrak{m}_k$. Note that in this case there is a trade-off between time and memory: On one end one can improve the performance of the F_4 algorithm by implementing the SIMPLIFY function and storing (part of) $F_j, \tilde{F}_j$. Whereas on the other end one can reduce the

memory usage of F_4 algorithm by ignoring the SIMPLIFY function and discard $F_j, \tilde{F}_j$ in each iteration, at the cost of increasing time complexity to compute Gröbner bases.

5.2 M4GB Reduction

Suppose that we want to reduce a nonzero polynomial $f \in \mathcal{R}$ by a nonempty intermediate basis $G \subset \mathcal{R}$. Let $\mathfrak{t}$ be a term of f and let $g \in G$ such that $\mathsf{LM}(g)$ divides the monomial of $\mathfrak{t}$. The standard reduction technique directly constructs a reductor $r = \mathfrak{t}/\mathsf{LT}(g) \cdot g$ and replaces f by $f - r$ to eliminate $\mathfrak{t}$ from f. It is possible that the polynomial r may be used in future iterations as a reductor. Thus in order to avoid recomputing r from g, one can store r as an element of the intermediate basis G. An approach to efficiently store polynomials in the intermediate basis G, based on Remark 5.1, is to ensure that each polynomial in G is tail-reduced w.r.t. G itself. To achieve this, we develop a new reduction algorithm that prioritizes tail-reduction of the newly constructed reductor over the reduction of f. We refer to this algorithm as the M4GBREDUCTION and it is described in Algorithm 5.1.

The algorithm iterates over each term $\mathfrak{t}$ of a polynomial f and checks the reducibility of $\mathfrak{t}$. In case $\mathfrak{t}$ is reducible then to obtain the reductor that eliminates $\mathfrak{t}$, the function GETREDUCTORBASIC is called (see Algorithm 5.2) with input parameter G and $\mathfrak{t}$. Suppose that $\mathfrak{t} = \mathfrak{c} \cdot \mathfrak{m}$ where $\mathfrak{c} \in \mathbb{F}, \mathfrak{m} \in \mathsf{M}(\mathcal{R})$ and assume there exists a $g \in G$ such that $\mathsf{LM}(g) \mid \mathfrak{m}$. The function GETREDUCTORBASIC returns a reductor for $\mathfrak{t}$ in two separate cases. The first case is when there exists a $g \in G$ such that $\mathsf{LM}(g) = \mathfrak{m}$, otherwise in the second case there is a g such that $\mathsf{LM}(g)$ strictly divides $\mathfrak{m}$. In the former, GETREDUCTORBASIC directly returns g as a reductor while in the latter case it ensures that the newly constructed reductor $\mathfrak{t}/\mathsf{LT}(g) \cdot g$ for $\mathfrak{t}$ is tail-reduced. Note that reducing the tail of $\mathfrak{t}/\mathsf{LT}(g) \cdot g$ is equal to the reducing the polynomial $\ell = \mathfrak{t}/\mathsf{LT}(g) \cdot \mathsf{Tail}(g)$. The function GETREDUCTORBASIC then calls the M4GBREDUCTION with input G and ℓ. The polynomial $\mathfrak{t} + \ell'$ is the tail-reduced reductor for $\mathfrak{t}$, where ℓ' is the polynomial output of M4GBREDUCTION(G, ℓ).

Example 5.3. *Let $\mathcal{R} = \mathbb{F}_2[x_1, x_2, x_3, x_4]$ we use the degree-reverse lexicographic (degrevlex) ordering on the monomials of $\mathcal{R}$. Suppose that we want to compute the remainder after division of $f = x_1^2x_2^3 + x_1x_2^3x_4 + x_1x_3^3 + x_1^3x_4 + x_2x_3^2 + x_4^2 \in \mathcal{R}$ by $G = \{g_1, g_2, g_3\} \subset \mathcal{R}$ where*

$$\begin{aligned} g_1 &= x_1^2x_2^3 + x_1x_3^3 + x_4^2, \\ g_2 &= x_2^3x_4 + x_2x_3 + x_3 + 1, \\ g_3 &= x_1x_2x_3 + x_1x_3. \end{aligned}$$

Note that the polynomials in G are already tail-reduced.

1. $(\mathfrak{t} = x_1^2x_2^3)$ *Since $\mathfrak{t}$ is reducible by G and $\mathsf{LM}(g_1) = x_1^2x_2^3$, then the function* GETREDUCTORBASIC$(G, \mathfrak{t})$ *is called and immediately returns (g_1, G). The polynomial h is equal to*

$$h = h - \mathfrak{t}/\mathsf{LT}(g_1) \cdot \mathsf{Tail}(g_1) = x_1x_3^3 + x_4^2.$$

2. $(\mathfrak{t} = x_1x_2^3x_4)$ *Since $\mathsf{LT}(g_2) \mid \mathfrak{t}$ then the function* GETREDUCTORBASIC$(G, \mathfrak{t})$ *is again called in this iteration. At this step, we have a case where $\mathsf{LM}(g_2)$*

Input: A set $G \subset \mathcal{R}$
Input: A polynomial $f \in \mathcal{R}$
Output: An updated basis $G' \supseteq G$ with $\langle G' \rangle = \langle G \rangle$
Output: Either 0 or a full-reduced nonzero polynomial $h = f - \sum_i g_i p_i \in \mathcal{R}$ where $g_i \in G', p_i \in \mathcal{R}$ such that $\mathsf{LM}(g_i p_i) \leq \mathsf{LM}(f)$ whenever $g_i p_i \neq 0$

```
1 h ← 0
2 G' ← G
3 forall t ∈ T(f) do
4     if ∃g ∈ G' : LT(g) | t then
          /* LM(g) = LM(t), g is tail-reduced */
5         (G', g) ← GetReductorBasic(G', t)
6         h ← h − (t/LT(g)) · Tail(g)
7     else
8         h ← h + t
9 return (G', h)
```

Algorithm 5.1: M4GBReduction

strictly divides the monomial of t. *Thus, we construct the polynomial* $\ell = \mathsf{t}/\mathsf{LT}(g_2) \cdot \mathsf{Tail}(g_2) = x_1x_2x_3 + x_1x_3 + x_1$ *and call* M4GBReduction(G, ℓ).

2.1. $(\mathsf{t} = x_1x_2x_3)$ *Since* $\mathsf{LT}(g_3) | \mathsf{t}$ *and* $\mathsf{LM}(g_3) = x_1x_2x_3$ *then the function* GetReductorBasic(G, t) *returns* (g_3, G). *Thus,*

$$h = h - (\mathsf{t}/\mathsf{LT}(g_3)) \cdot \mathsf{Tail}(g_3) = x_1x_3.$$

2.2. $(\mathsf{t} = x_1x_3)$ *Since* t *is not reducible by* G, *then* $h = h + \mathsf{t} = 0$.

2.3. $(\mathsf{t} = x_1)$ *Similarly in this case* t *is not reducible by* G, *and finally*

$$h = h + \mathsf{t} = x_1.$$

The newly constructed reductor g_4 *is equal to*

$$g_4 = \mathsf{t} + \ell' = x_1x_2^3x_4 + x_1.$$

The intermediate basis G *now consists of the polynomials* $G = \{g_1, g_2, g_3, g_4\}$ *and* GetReductorBasic *returns* (G, g_4). *The polynomial* h *is now equal to*

$$h = h - (\mathsf{t}/\mathsf{LT}(g_4)) \cdot \mathsf{Tail}(g_4) = x_1x_3^3 + x_4^2 + x_1.$$

3. At this stage, no remaining term of f *is reducible by* G. *One can verify that* M4GBReduction *eventually returns* (G, h) *where*

$$h = x_1^3x_4 + x_2x_3^2 + x_1$$

and $G = \{g_1, g_2, g_3, g_4\}$.

Input: A set $G = \{g_1, \ldots, g_s\} \subset \mathcal{R}$
Input: A term $\mathfrak{t} \in \mathsf{T}(\mathcal{R})$
Output: An updated basis $G' \supseteq G$ with $\langle G' \rangle = \langle G \rangle$
Output: A tail-reduced polynomial $h \in G'$ such that $\mathsf{LM}(h) = \mathsf{LM}(\mathfrak{t})$

```
1  G' ← G
2  if ∃g ∈ G' : LM(g) = LM(t) then
3  |  h ← g
4  else
5  |  Select g ∈ G' such that LM(g) | LM(t)
6  |  ℓ ← t/LT(g) · Tail(g)
7  |  (G', ℓ') ← M4GBReduction(G', ℓ)
8  |  g_{s+1} ← t + ℓ'
9  |  G' ← G' ∪ g_{s+1}
10 |  h ← g_{s+1}
11 return (G', h)
```

Algorithm 5.2: GetReductorBasic

5.3 M4GB Algorithm

In this section we give the description of the M4GB algorithm. It is an extension of the Buchberger's algorithm with the specific aim to store and process polynomials only in tail-reduced form. In the remaining of this chapter, the intermediate basis in M4GB algorithm is denoted by M.

5.3.1 M4GB Invariant

The main difference in the strategy of maintaining intermediate basis in the M4GB algorithm compared to the Buchberger's algorithm described in Algorithm 4.2 is that the intermediate basis M in M4GB is not kept minimal, i.e., for some $f \in M$ there exists $g \in M$ such that $\mathsf{LM}(g)$ strictly divides $\mathsf{LM}(f)$. The reason for keeping redundant elements in M is to avoid their recomputation as reductors in the subsequent iterations. In addition to that, M4GB algorithm sets a restriction that there exists no two distinct nonzero polynomials in M with equal leading monomial. The rationale of this approach is based on Remark 5.2.

However, there are two problems with such strategy. Firstly, applying Gebauer-Möller installation on M would remove the redundant polynomials from M, thus contradicting our approach. Secondly, if we let M to be non-minimal and at the same time we did not employ the Gebauer-Möller installation, then the algorithm would generate many useless critical pairs. To overcome this situation, we introduce a set of monomials $L \subset \mathsf{M}(\mathcal{R})$. The purpose of the set L is to indicate the elements of M that constitute a minimal intermediate basis of the ideal. In addition to that, the set L replaces the role of the intermediate basis M inside the Gebauer-Möller installation and the reducibility of a polynomial by the set M can be determined more efficiently using L. We summarize this strategy in the following notion.

Definition 5.4 (M4GB Invariant)**.** *The pair* (L, M) *where* $L \subset \mathsf{M}(\mathcal{R})$ *and* $M \subset \mathcal{R}$ *is said to satisfy the* M4GB *invariant if*

1. *For all* $f, g \in M$, $\mathsf{LM}(f) = \mathsf{LM}(g)$ *implies that* $f = g$. *For any monomial* $\mathfrak{m} \in \mathsf{LM}(M)$, *we define* $M[\mathfrak{m}]$ *to be the polynomial* $g \in M$ *such that* $\mathsf{LM}(g) = \mathfrak{m}$. *Moreover, for any nonempty subset* $L \subseteq \mathsf{LM}(M)$ *we also define* $M[L] = \{f \in M : \mathsf{LM}(f) \in L\}$.
2. *The set* L *is a subset of* $\mathsf{LM}(M)$,
3. *The ideal* $\langle M \rangle$ *is equal to* $\langle M[L] \rangle$,
4. *For all* $f \in M$, *there exists* $\mathfrak{m} \in L$ *such that* $\mathfrak{m} \mid \mathsf{LM}(f)$,
5. *For all* $f \in M$, *the polynomial* f *is not tail-reducible by* L.

5.3.2 Generalization of M4GB Reduction

In the previous section, we have introduced a new approach to perform polynomial reduction with the M4GBReduction. In the actual M4GB algorithm, we generalize the M4GBReduction to polynomials of the form $\mathfrak{t} \cdot f$ where $\mathfrak{t} \in \mathsf{T}(\mathcal{R})$ and $f \in \mathcal{R}$. This generalization allows reduction on $\mathfrak{t} \cdot f$ without the explicit construction of the polynomial $\mathfrak{t} \cdot f$ itself, a strategy that is not employed by both the Buchberger and the F_4 algorithm. The resulting algorithm is called the MulFullReduce and it is described in Algorithm 5.3.

Input: A pair of set (L, M) where $L \subset \mathsf{M}(\mathcal{R}), M \subset \mathcal{R}$ satisfying M4GB invariant
Input: A term $\mathfrak{t} \in \mathsf{T}(\mathcal{R})$
Input: A polynomial $f \in \mathcal{R}$
Output: An updated intermediate basis $M' \supseteq M$ with $\langle M' \rangle = \langle M \rangle$
Output: Either 0 or a full-reduced nonzero polynomial $h = \mathfrak{t}f - \sum_i g_i p_i \in \mathcal{R}$, where $g_i \in M', p_i \in \mathcal{R}$ such that $\mathsf{LM}(g_i p_i) \leq \mathsf{LM}(\mathfrak{t}f)$ whenever $g_i p_i \neq 0$

```
1  M' ← M
2  h ← 0
3  forall 𝔰 ∈ T(f) do
4      𝔯 ← 𝔱 · 𝔰
5      if ∃𝔪 ∈ L : 𝔪 | 𝔯 then
6          (M', g) ← GetReductor(L, M', 𝔯)
7          h ← h − (𝔯/LT(g)) · Tail(g)
8      else
9          h ← h + 𝔯
10 return (M', h)
```

Algorithm 5.3: MulFullReduce Algorithm.

The MulFullReduce function generalizes the M4GBReduction by performing divisibility check and the reduction on the terms $\mathfrak{t} \cdot \mathfrak{s}$ for all $\mathfrak{s} \in \mathsf{T}(f)$. Another difference with the M4GBReduction is that the reducibility test in MulFullReduce uses the set L instead of the intermediate basis M, which minimizes the number of divisibility checks. The function GetReductor is an adjustment of the GetReductorBasic by incorporating the set L and M.

Input: A pair of set (L, M) where $L \subset \mathsf{M}(\mathcal{R}), M \subset \mathcal{R}$ satisfying M4GB invariant
Input: A term $\mathfrak{t} \in \mathsf{T}(\mathcal{R})$
Output: An updated intermediate basis $M' \supseteq M$ with $\langle M' \rangle = \langle M \rangle$
Output: A tail-reduced reductor $h \in \langle M \rangle$ with $\mathsf{LM}(h) = \mathsf{LM}(\mathfrak{t})$

1 $M' \leftarrow M$
2 **if** $\mathsf{LM}(\mathfrak{t}) \in \mathsf{LM}(M')$ **then**
3 $\quad h \leftarrow M'[\mathsf{LM}(\mathfrak{t})]$
4 **else**
5 $\quad$ Select $\mathfrak{m} \in L$ such that $\mathfrak{m} \mid \mathfrak{t}$
6 $\quad r \leftarrow M'[\mathfrak{m}]$
7 $\quad (M', \ell') \leftarrow \textsc{MulFullReduce}(L, M', \mathfrak{t}/\mathsf{LT}(r), \mathsf{Tail}(r))$
8 $\quad h \leftarrow \mathfrak{t} + \ell'$
9 $\quad M' \leftarrow M' \cup \{h\}$
10 **return** (M', h)

Algorithm 5.4: GetReductor Algorithm.

Similar to the MulFullReduce function, it uses the set L to find a polynomial in M that will be used to construct the reductor.

When a term in $\mathfrak{t}f$ is reducible by M' (line 5-7 of Algorithm 5.3), GetReductor is called which would possibly followed by another call to MulFullReduce to perform tail-reduction on a newly constructed reductor (line 7 of Algorithm 5.4). One then asks if the initial call to the MulFullReduce eventually terminates. Suppose that we call $\textsc{MulFullReduce}(L, M, \mathfrak{t}, f)$ and let $\mathfrak{s} \in \mathsf{T}(f)$ such that there exists $\mathfrak{m} \in L$, $\mathfrak{m} \mid \mathfrak{r} = \mathfrak{t}\mathfrak{s}$ and $\mathsf{LM}(\mathfrak{r}) \notin \mathsf{LM}(M')$. Then GetReductor will call $\textsc{MulFullReduce}(L, M', \mathfrak{r}/\mathsf{LT}(r), \mathsf{Tail}(r))$. Assuming $\mathsf{Tail}(r) \neq 0$, the important remark here is that

$$\mathsf{LM}(\mathfrak{r}/\mathsf{LT}(r) \cdot \mathsf{Tail}(r)) = \mathsf{LM}(\mathsf{Tail}(\mathfrak{r}/\mathsf{LT}(r) \cdot r)) < \mathfrak{r} \leq \mathsf{LM}(\mathfrak{t}f).$$

Since by definition a monomial ordering $<$ is a well-ordering, (equivalently every strictly decreasing sequence of monomials eventually terminates). Therefore a call to $\textsc{MulFullReduce}(L, M, \mathfrak{t}, f)$ always terminates.

5.3.3 Computation of S-Polynomials

The function MulFullReduce can easily be used to avoid unnecessary step in the computation of S-polynomials. Recall that the S-polynomial of $f, g \in \mathcal{R} \backslash \{0\}$ is defined as

$$\mathsf{Spoly}(f, g) = \frac{u}{\mathsf{LT}(f)} \cdot f - \frac{u}{\mathsf{LT}(g)} \cdot g$$

where $u = \mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(g))$. The notion of S-polynomial was introduced to produce cancellation of leading terms. Despite this cancellation is known in advance, Buchberger algorithm and F_4 algorithm still perform explicit computation of S-polynomials (in the F_4 algorithm, an S-polynomial is computed by substraction of two rows in the coefficient matrix that correspond to a critical pair). Recall that by Remark 4.4 the S-polynomial $\mathsf{Spoly}(f, g)$ can be expressed as

$$\mathsf{Spoly}(f, g) = \frac{u}{\mathsf{LT}(f)} \cdot \mathsf{Tail}(f) - \frac{u}{\mathsf{LT}(g)} \cdot \mathsf{Tail}(g).$$

The M4GB algorithm computes the reduction of S-polynomials by taking the advantage that the cancellation of leading terms is known in advance. It first computes the reduction of the polynomial $\frac{u}{\mathsf{LT}(f)} \cdot \mathsf{Tail}(f)$ followed by the reduction of $\frac{u}{\mathsf{LT}(g)} \cdot \mathsf{Tail}(g)$. These reductions are done by calling the function MulFullReduce with parameters $(L, M, u/\mathsf{LT}(f), \mathsf{Tail}(f))$ and $(L, M, -u/\mathsf{LT}(g), \mathsf{Tail}(g))$ respectively. Finally, the resulting polynomial outputs are added.

5.3.4 The M4GB Algorithm

The complete description of the M4GB algorithm is given in Algorithm 5.5. During the initialization phase and for every new polynomial found in the ideal as a result of the reduction of S-polynomial, the algorithm calls the function UpdateReduce. Suppose that we call it with parameters (L, M, P, f). The goal of the UpdateReduce function is to perform tail-reduction on polynomials in the intermediate basis M using f. The tail-reduction of polynomials in M is achieved in two phases. The first phase, described in line 3-7 of Algorithm 5.6, is a collection of all necessary reductors constructed from f that will be used to eliminate terms in the tail of polynomials in M. The constructed reductors are stored in the set H. In addition to the reductors of polynomials in M, the same step also constructs reductors to perform tail-reduction on polynomials in H itself. This can be seen on the set Q that keeps updated during the iteration. This will be followed by the second phase, which is the actual tail-reduction of polynomials in M and H. Finally, UpdateReduce calls the Gebauer-Möller installation as described in Algorithm 4.3.

Input: A finite nonempty subset $F \subset \mathcal{R}$
Output: A Gröbner basis G of $\langle F \rangle$

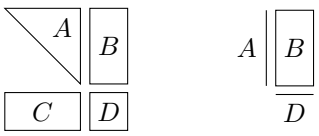
```
1  L ← {}
2  M ← {}
3  P ← {}
4  forall f ∈ F do
5  |  (M, f) ← MulFullReduce(L, M, 1, f)
6  |  (L, M, P) ← UpdateReduce(L, M, P, f)
7  while P ≠ {} do
8  |  (f_LM, g_LM) ← Select(P)
9  |  P ← P \ {(f_LM, g_LM)}
10 |  (f, g) ← (M[f_LM], M[g_LM])
11 |  u ← LCM(f_LM, g_LM)
12 |  (M, h1) ← MulFullReduce(L, M, u/LT(f), Tail(f))
13 |  (M, h2) ← MulFullReduce(L, M, u/LT(g), Tail(g))
14 |  h ← h1 − h2
15 |  if h ≠ 0 then
16 |  |  (L, M, P) ← UpdateReduce(L, M, P, h)
17 G ← M[L]
18 return G
```

Algorithm 5.5: M4GB Algorithm.

Input: A pair of set (L, M) where $L \subset \mathsf{M}(\mathcal{R}), M \subset \mathcal{R}$ satisfying M4GB invariant
Input: A set of critical pairs $P \subset \mathsf{M}(\mathcal{R}) \times \mathsf{M}(\mathcal{R})$
Input: A polynomial $f \in \langle M \rangle \setminus M$
Output: An updated L', M', satisfying M4GB invariant and $\langle M[L] \rangle = \langle M'[L'] \rangle$
Output: An updated set of critical pairs P'

$M' \leftarrow M$
$L' \leftarrow L$
// The set H contains polynomials to add to M', to maintain M4GB invariant while adding f
$H \leftarrow \{\mathsf{LC}(f)^{-1} \cdot f\}$
// The set Q contains monomials that needs to be processed
$Q \leftarrow \mathsf{M}(\mathsf{Tail}(M' \cup H)) \setminus \mathsf{LM}(H)$
while $\exists \mathfrak{m} \in Q : \mathsf{LM}(f) \mid \mathfrak{m}$ **do**
    $\mathfrak{m}' \leftarrow \max\{\mathfrak{m} \in Q : \mathsf{LM}(f) \mid \mathfrak{m}\}$
    $(M', h) \leftarrow \textsc{MulFullReduce}(L', M', \mathfrak{m}'/\mathsf{LT}(f), \mathsf{Tail}(f))$
    $H \leftarrow H \cup \{\mathfrak{m}' + h\}$
    $Q \leftarrow \mathsf{M}(\mathsf{Tail}(M' \cup H)) \setminus \mathsf{LM}(H)$
while $H \neq \{\}$ **do**
    Select $h \in H$ such that $\mathsf{LM}(h) = \min(\mathsf{LM}(H))$
    $H \leftarrow \{g - ch : g \in H, c$ is the coefficient of LM(h) in Tail(g)$\}$
    $M' \leftarrow \{g - ch : g \in M', c$ is the coefficient of LM(h) in Tail(g)$\}$
    $M' \leftarrow M' \cup \{h\}$
    $H \leftarrow H \setminus \{h\}$
$(L', P') \leftarrow \textsc{Update}(L', P, \mathsf{LM}(f))$
return (L', M', P')

Algorithm 5.6: UpdateReduce Algorithm.



(a) *F_4 matrix* **(b)** M4GB*'s M*

Columns represent monomials, with non-reducible monomials last. Rows represent polynomials, with S-polynomial halves at the bottom (CD) and reductor polynomials at the top (AB). We remark that A in M4GB *is a diagonal submatrix, represented as rowlabels of B.*

Figure 7: *Visualization of F_4's matrix and* M4GB*'s M*

5.4 Comparison with the F_4 Algorithm

In this section we shall discuss a comparison between M4GB algorithm and the F_4 algorithm.

The F_4 algorithm achieves the reduction of S-polynomials by translating the reduction of many S-polynomials to the computation of (reduced)-row echelon form of the coefficient matrix. The M4GB algorithm performs addition and scalar multiplication on tail of polynomials, which can be efficiently implemented as scalar multiplication and addition of coefficient vectors. Both algorithms keep track of prior reduced polynomials to speed up computations in the next iterations. The F_4 algorithm achieved this using the SIMPLIFY function, which requires the algorithm to store all previously constructed coefficient matrices and their corresponding (reduced)-row echelon forms. In the M4GB algorithm, this is clearly achieved by the set M that not only contains the tail-reduced intermediate basis but also tail-reduced multiples thereof that have been used as reductors.

It should be noted that the M4GB algorithm has a more native description instead of translating desired computations into an external linear algebra computation. However, there are more important differences between the M4GB and the F_4 algorithm that would indicate that the former is more computational and memory efficient than the latter.

Firstly, the M4GB algorithm reduces one critical pair at a time in contrast to the F_4 algorithm that selects many critical pairs. In fact, the best known selection strategy for the F_4 algorithm is to choose all critical pairs of the smallest degree (normal selection strategy). This seems to be the most efficient even if more critical pairs are selected than necessary, because it reduces the overhead associated in translating to and from coefficient matrices. Also reducing the number of critical pairs may not significantly decrease the matrix size anyway. Unfortunately, when a relatively small number of these critical pairs would be sufficient, the F_4 algorithm wastes a significant amount of computation and memory. This is easily seen in the complexity graph for a growing set of problems where there are significant "jumps" whenever the degree of regularity increase (see Figure 8). By operating on a single critical pair (or small batches) at a time, the M4GB algorithm does not have this disadvantage as can be seen from our experiments.

Secondly, the M4GB algorithm operates on coefficient vectors over non-

reducible monomials, which in effect eliminates unnecessary computations and memory use involving reducible monomials. In Figure 7 we have visualized the M4GB's M against matrices used by the F_4 algorithm to showcase this benefit. One can directly see that the F_4 algorithm works with the upper-triangular matrix A as well as larger matrices C and D. Note that for the M4GB's M, instead of representing A as a diagonal matrix, we represent A as a single column of leading terms (instead of coefficients), or equivalently as the labels for the coefficient rows in B. It is well known that matrices generated by the F_4 algorithm have special structure and most row pivots are known in advance, namely those in A related to the reductor polynomials. Linear algebra software can take advantage of this special structure, but inherently must spent computation and memory in keeping track of coefficients related to reducible monomials. In contrast, whenever a reducible terms arises, the M4GB algorithm acts on this and reduces it without ever having to store the reducible term in the result.

5.5 Correctness

We shall now discuss the correctness of the M4GB algorithm to compute a Gröbner bases. We limit ourselves to proving the correctness of full-reduction in the initial basis of the ideal and the S-polynomials since the algorithm itself performs the same steps as Buchberger's algorithm for any selection strategy and Update function.

The MulFullReduce algorithm computes $\mathfrak{t} \cdot f$ by multiplying each term $\mathfrak{s} \in \mathsf{T}(f)$ by $\mathfrak{t}$ and if $\mathfrak{t}\cdot\mathfrak{s}$ is non-reducible by L, the result is added to h. Otherwise, when the term $\mathfrak{r} = \mathfrak{t} \cdot \mathfrak{s}$ is reducible by L it will immediately reduce it with a reductor $g \in M$ with $\mathsf{LM}(g) = \mathsf{LM}(\mathfrak{r})$ retrieved by calling GetReductor:

$$h \leftarrow h + \mathfrak{r} - (r/\mathsf{LT}(g)) \cdot g = h - (r/\mathsf{LT}(g)) \cdot \mathsf{Tail}(g).$$

Assuming correctness of GetReductor we have that (L, M) satisfies the M4GB invariant at every step and that any such g is tail-reduced by L itself*. Thus $\mathsf{Tail}(g)$ is full-reduced by L and the resulting h is always full-reduced by L.

For the correctness of GetReductor, if for the input term $\mathfrak{t}$ there exists a polynomial $g \in M$ with $\mathsf{LM}(g) = \mathsf{LM}(\mathfrak{t})$ then it immediately returns g which is tail-reduced by L due to M4GB invariant. Otherwise, it selects a monomial $\mathfrak{m} \in L$ such that $\mathfrak{m} \mid \mathsf{LM}(\mathfrak{t})$ and retrieves the unique $r \in M$ with $\mathsf{LM}(r) = \mathfrak{m}$ (which always exists since $L \subset \mathsf{LM}(M)$). It then computes

$$\mathfrak{t}/\mathsf{LT}(r) \cdot r = \mathfrak{t} + \mathfrak{t}/\mathsf{LT}(r) \cdot \mathsf{Tail}(r)$$

in tail-reduced form $\mathfrak{t} + \ell'$ by calling MulFullReduce to compute the full-reduced tail ℓ'. Note that the set $M \cup \{\mathfrak{t}+\ell'\}$ still satisfies the M4GB invariant.

We will now discuss the correctness of UpdateReduce. It first updates the set M so that all polynomials $g \in M$ are not tail-reducible by $L \cup \{\mathsf{LM}(f)\}$. Starting with $H = \{\mathsf{LC}(f)^{-1} \cdot f\}$, it will repeatedly insert $\mathfrak{m}'/\mathsf{LT}(f) \cdot f$ into H for the largest monomial $\mathfrak{m}' \in \mathsf{M}(\mathsf{Tail}(M \cup H)) \setminus \mathsf{LM}(H)$ that is reducible

* We also remark that L remains unchanged, i.e., a call to GetReductor does not cause h to not be tail-reduced w.r.t. L.

by f that still needs to be handled. Note that for any such $\mathfrak{m}'$, it will call MulFullReduce which only calls GetReductor$(L, M, \mathfrak{r})$ for $\mathfrak{r} < \mathfrak{m}'$. The sequence of chosen monomials $\mathfrak{m}'$ in line 4 of Algorithm 5.6 is monotonic decreasing and thus finite, as any polynomial h in line 5 has $\mathsf{LM}(h) < \mathfrak{m}'$. After this procedure, at the beginning of step 8, for any monomial $\mathfrak{m}' \in \mathsf{M}(\mathsf{Tail}(M \cup H))$ reducible by f, there exists $h \in H$ with $\mathsf{LM}(h) = \mathfrak{m}'$. At the same time, every $g \in M \cup H$ are tail-reduced by L due to the M4GB invariant and the correctness of MulFullReduce. Therefore, after processing all $h \in H$ in line 9-13 we have that no monomial $\mathfrak{m}' \in \mathsf{M}(\mathsf{Tail}(M))$ is reducible by $L \cup \mathsf{LM}(f)$. Since eventually $\mathsf{LC}(f)^{-1} \cdot f \in M$ due to the step in line 1 and 12, the M4GB invariant will be broken during the steps in line 8-13, however this is fixed when $\mathsf{LM}(f)$ is inserted to L by the Update function in line 14.

For the correct application of Gebauer-Möller installation, let $g_1, g_2, \ldots$ be the sequence of polynomials passed to the UpdateReduce function for M4GB. Note that Gebauer-Möller installation may remove an element g_i from the intermediate basis but keep the critical pair (g_i, h) for some polynomial h in the ideal. It suffices to use a proper references (integer indexing or leading monomial) in the set of critical pairs. Since the polynomials $g_1, g_2, \ldots$ are inserted to M, never removed, never replaced, and only further tail-reduced by UpdateReduce, in M4GB we represent critical pairs by leading monomials and hence Algorithm 5.5 extracts the correct polynomials in line 10.

Lastly, it remains to verify that the steps in line 11-14 of Algorithm 5.5 indeed compute the full-reduction of the S-polynomial of f and g. The full-reduction of $\mathsf{Spoly}(f, g)$ can be computed as the differences of the full-reduction of $(u/\mathsf{LT}(f)) \cdot \mathsf{Tail}(f)$ and $u/\mathsf{LT}(g) \cdot \mathsf{Tail}(g)$ where $u = \mathsf{LCM}(\mathsf{LM}(f), \mathsf{LM}(h))$, which is computed in line 12-13 of Algorithm 5.5 since full-reduction distributes over polynomial addition for fixed basis. Let $G = \{f \in M : \mathsf{LM}(f) \in L\}$ is fixed and for any monomial u reducible by G, a fixed tail-reduced polynomial h is given by GetReductor. Let f_1, f_2 be any two polynomials with coefficients $c_1, c_2 \in \mathbb{F}$ for monomial u respectively. In the full-reduction of f_1, f_2 and $f_1 + f_2$ the only tail-reduced polynomial that affects the terms with monomial u is the identical corresponding h with $\mathsf{LM}(h) = u$. Without loss of generality, assume that h is monic, then $f_1 - c_1 h, f_2 - c_2 h$ and $(f_1 + f_2) - ch$ are full-reduced if and only if $c_1 + c_2 = c$, which implies that full-reduction distributes over polynomial addition for fixed tail-reduced reductors for reducible monomials.

5.6 Efficient Implementation

We provide a proof-of-concept implementation of the M4GB algorithm. The implementation is tailored for dense overdefined systems over a finite field. Note that an overdefined system of equations over a finite field typically have a unique solution over the base field. In particular, we evaluate our implementation against systems of polynomials equations that represent public-keys in multivariate public-key and digital signature algorithm. The source code of the M4GB algorithm, written in `C++11`, is available under the GPLv3 license at

`https://github.com/cr-marcstevens/m4gb`.

Our implementation of the algorithm supports degree-reverse lexicographic ordering, which is considered as a favorable ordering for Gröbner bases computation. It consists mainly of finite field, monomials, parser, and threadpool

implementations. It supports prime fields and even-characteristic finite fields of small size. To facilitate timing and memory comparison, we also provide convenient wrappers for FGb and OpenF4. The implementation of the M4GB algorithm is also configurable at compile-time to optimize the run-time performance and to produce more efficient compiled code. It automatically selects the most efficient data types based on the order of the finite field and the number of variables in the polynomial ring.

The M4GB algorithm performs computations on leading terms and tails separately and frequently retrieves polynomials by their leading monomial. The most efficient data structure for M is a hash-map, mapping the leading monomial to the leading coefficient and the tail.

Operations on the tails of polynomials in M are scalar multiplication and polynomial addition which, in the case of dense polynomials, are more efficient to compute if these tails are stored as coefficient vectors relative to a global list of non-reducible monomials in increasing order and are truncated by removing trailing zeros. The global list stores monomials up to the largest least common multiple of leading monomials in the set of critical pairs. It enables a simpler implementation that does not have to deal with ad-hoc column insertions but at the same time it allows a faster way to determine all reducible and non-reducible monomials using an approach comparable to the sieve of Eratosthenes [Sho08, pg. 115]. The monomials are represented as machine-size integers rather than as exponent vectors. We implemented a fast order-preserving encoder/decoder for monomials that utilizes small look-up tables that fit CPU cache memory.

For system of equations that represent the public-key of multivariate cryptosystem, the order of the underlying finite field $\mathbb{F}_q$ is typically small with $q \leq 256$. We have chosen to implement finite field operations in a relatively simple way by using log and reverse-log lookup-tables, which remains performant at least up to $q \leq 65535$. For $q \leq 256$ we additionally use a two-dimensional array as a multiplication look-up table that easily fits into CPU cache memory. Furthermore, for prime fields with $q \leq 31$ we use a three-dimensional array as a multiply-and-add look-up table, i.e., the entry $A[i][j][k]$ stores the value $((i \cdot j) + k) \pmod q$ for all $i, j, k \in \mathbb{F}_q$.

5.7 Benchmark and Performance Comparison

We compared our implementation of the M4GB algorithm against existing state-of-the-art implementations of Gröbner bases algorithm: FGb [*], Magma version 2.20-6 [†], and OpenF4[‡]. While the first two are well-known implementations of Gröbner bases algorithm, the latter came to our attention when it was proposed as an optional package for SageMath [§]. Unlike Magma and FGb, OpenF4 is an open-source implementation of F_4 algorithm that uses SIMD CPU instructions which makes it an attractive competitor to existing closed-source implementation of Gröbner bases algorithm.

[*] Available at `http://www-polsys.lip6.fr/~jcf/FGb/C/index.html`.
[†] Available at `http://magma.maths.usyd.edu.au/magma/`.
[‡] Available at `https://github.com/nauotit/openf4`.
[§] See `https://trac.sagemath.org/ticket/18749`.

5.7.1 Machines

There are two different machines used for benchmarking purposes :

A) Dual Intel Xeon E5-2650 v3 @ 2.3 GHz with 128GB RAM and

B) Quad Intel Xeon E5-4640 @ 2.4 GHz with 132GB RAM.

The first machine has been used to run and compare the performance of M4GB, FGb, and OpenF4 while the comparison of M4GB with Magma was done in the second machine. Both machines are Non-Uniform Memory Access (NUMA) machine where memories are distributed accross different CPU chips. To ensure that the cost of process and memory transfers among CPU chips does not affect the experimental result, we enforce the processes to run in one CPU chip and its associated memory using `numactl` available in most UNIX system. We are also aware that the processors in both machines support Intel Hyper-Threading technology, and we ensure all processes run in the physical cores.

5.7.2 Testcases

The test cases used to benchmark and compare the chosen implementations of Gröbner bases algorithms consist of polynomial equation systems relating to multivariate public-key encryption and signature schemes. These polynomial systems are dense, quadratic, overdefined, and all coefficients are randomly chosen from the finite field. The public challenges to solve MQ problems* are based on three different finite fields: $\mathbb{F}_2, \mathbb{F}_{2^8}, \mathbb{F}_{31}$. Since $\mathbb{F}_{2^8}$ is not supported by FGb and $\mathbb{F}_2$ needs more specialized data structure for the polynomials, our benchmarks are limited to the field $\mathbb{F}_{31}$. The number of variables (n) and equations (m) are chosen such that the execution can be completed within a reasonable time.

Based on the number of variables and equations, there are two types of equation system used in the experiment. The first type is a strongly over-defined system of equations with $m = 2n$. The second type is a weakly overdefined system of equations with $m = n+1$. The first type of equation system represents equation system from multivariate public-key cryptography, while the second type of equation system is the result of hybrid-approach in solving underdefined system of equations that represent multivariate signature (see Chapter 6).

5.7.3 Results

Our benchmarking results for $m = 2n$ and $m = n + 1$ are listed in the Table 5 and Table 6 respectively. In order to allow comparisons in one single graph and table, the runtime and memory usage of Magma had to be projected. The actual results of comparison between Magma and M4GB is available in Table 8 and Table 9. Note that in light of the large ratios between the performance of Magma and the performance of M4GB, the margin of error caused by the projection for Magma should be insignificant. For the same reason and by taking into account the time to complete the executions, it was not worth repeating the experiments many times for the same parameters to obtain more accurate expected time complexities.

* `https://www.mqchallenge.org`

		Total CPU time (sec)			
n	m	OpenF4	FGb	Magma (projected)	M4GB
20	40	206	470	232.17	57
21	42	472	1002	500.26	170
22	44	1145	3118	1616.73	424
23	46	2274	6849	3184.82	1060
24	48	10293	64700	31167.61	2556
25	50	-	151653	77678.58	5575
26	52	-	360055	183628.74	15517
27	54	-	767543	409451.87	46548
		Memory (MB)			
20	40	4240	112	361.84	73
21	42	6640	165	577.34	121
22	44	14368	525	853.84	226
23	46	26135	918	1324.16	395
24	48	161945	1561	8872.94	663
25	50	-	2765	19718.78	1471
26	52	-	4607	25197	3328
27	54	-	8180	39844.84	6799

Table 5: *Performance comparison for systems with n variables and $m = 2n$ equations over $\mathbb{F}_{31}$. The corresponding graph is given in Figure 8.*

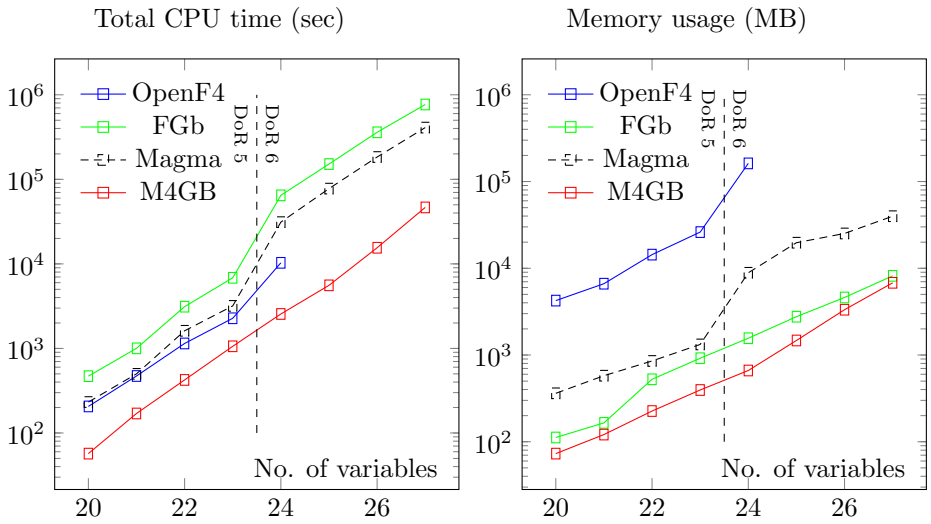


Figure 8: *The graph of CPU time and memory usage (log-y scale) with n variables and $m = 2n$ equations over $\mathbb{F}_{31}$. The vertical dashed line separates the parameter with different degree of regularity (DoR). The corresponding data is described in Table 5.*

		Total CPU time (sec)			
n	m	OpenF4	FGb	Magma (projected)	M4GB
10	11	2.99	5	3.29	0.98
11	12	8.73	21	11.172	2.6
12	13	36.76	134	59.08	13.92
13	14	172.49	642	286.4	58.18
14	15	1258	5850	2810.75	393.19
15	16	7225	36361	17265.5	2424
		Memory (MB)			
10	11	101	33	32.09	17
11	12	341	50	64.12	16
12	13	1463	112	113.59	31
13	14	7622	323	281.53	74
14	15	33460	1098	1104	250
15	16	117396	4118	3320	837

Table 6: *Performance comparison for systems with n variables and $m = n + 1$ equations over $\mathbb{F}_{31}$. The corresponding graph is given in Figure 9.*

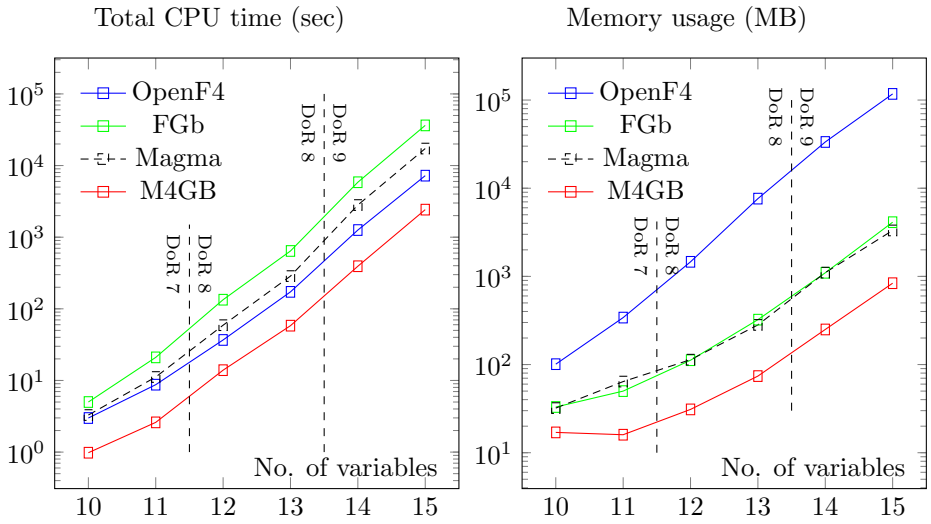


Figure 9: *The graph of CPU time and memory usage (log-y scale) with n variables and $m = n+1$ equations over $\mathbb{F}_{31}$. The vertical dashed line separates the parameter with different degree of regularity (DoR). The corresponding data is described in Table 6.*

5.7.4 Observations and Conclusions

In Table 5 one can see a clear trade-off between the CPU time and the memory usage among OpenF4, FGb, and Magma. OpenF4 performs best in terms of speed followed by Magma and FGb. However, FGb has the least memory usage followed by Magma and OpenF4. The efficiency of time execution of OpenF4 is likely due to its usage of SIMD instructions to perform Gaussian elimination. In terms of CPU time, OpenF4 performs 1.05 up to 3 times faster than Magma. However, its memory usage is at least 11 times more than Magma, which does not allow us to run OpenF4 for $n \geq 25$.

In comparison to OpenF4 (which is the fastest implementation among OpenF4, FGb, and Magma), our implementation of M4GB runs 2.15 and up to 4 times faster and this puts M4GB as the fastest one. In comparison with FGb (which uses the least memory among OpenF4, FGb, and Magma), M4GB uses less memory by a factor between 1.2 and 2.35 and this puts M4GB as an implementation with the least memory usage.

The benchmarking results for $m = n+1$ in Table 6 shows the same ordering in terms of CPU time performance, that is M4GB is the fastest followed by OpenF4, Magma, and FGb. In terms of memory usage, M4GB is also the most efficient one followed by both FGb and Magma which are comparably similar, and OpenF4. M4GB performs 2.6 up to 3.3 faster than OpenF4 and uses less memory by a factor of 1.9 up to 4.4 compared to Magma and FGb.

In order to understand the overall efficiency, we also computed the ratio of time-memory product of other implementations relative to the time-memory product of M4GB in Table 7. For the systems with n variables and $2n$ equations, OpenF4 has the highest ratio, ranging from 141.94 ($n = 23$) to 983.64 ($n = 24$), mainly due to its high memory usage. This is followed by Magma with ratio ranging from 10.07 ($n = 23$) to 186.77 ($n = 25$). Lastly, FGb has overall the smallest ratio ranging from 8.04 ($n = 21$) to 59.6 ($n = 24$). For systems with $n+1$ equations and n variables, OpenF4 also has the highest ratio from 18.13 ($n = 10$) up to 428.22 ($n = 14$). This is followed by FGb, with ratio ranging from 9.9 ($n = 10$) to 73.8 ($n = 15$). Lastly, Magma has the closest ratio to M4GB ranging from 6.34 ($n = 10$) to 31.57 ($n = 14$).

n	m	FGb	Magma	OpenF4
20	40	12.65	20.19	209.91
21	42	8.04	14.04	152.36
22	44	17.08	14.40	171.68
23	46	15.02	10.07	141.94
24	48	59.6	163.19	983.64
25	50	51.13	186.77	-
26	52	32.12	89.60	-
27	54	19.84	51.55	-

n	m	Magma	FGb	OpenF4
10	11	6.34	9.9	18.13
11	12	17.22	25.24	71.56
12	13	15.55	34.78	124.63
13	14	18.73	48.16	305.37
14	15	31.57	65.34	428.22
15	16	28.25	73.8	418.05

Table 7: *The ratio of time-memory product of Magma, FGb, and OpenF4 relative to the time-memory product of* M4GB *for systems with $m = 2n$ equations (left) and $m = n + 1$ equations (right) where n is the number of variables. The corresponding graph is given in Figure 10.*

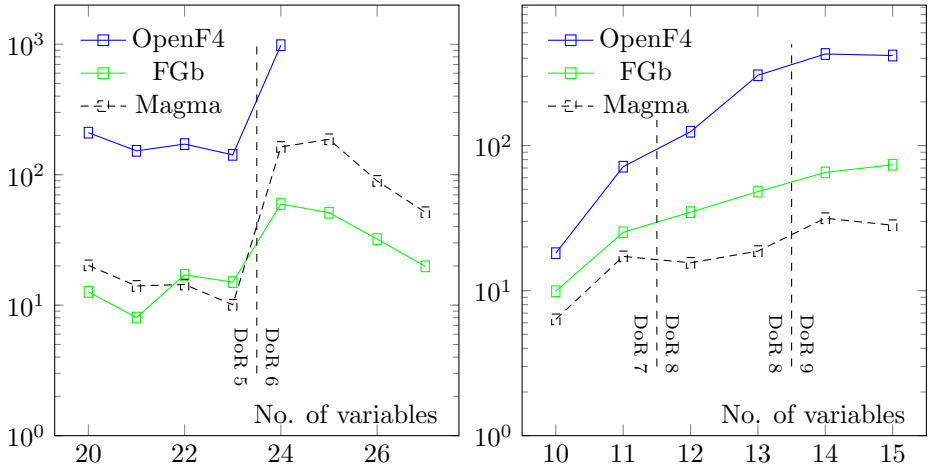


Figure 10: *The ratio of time-memory product of Magma, FGb, and OpenF4 (log-y axis) relative to the time-memory product of* M4GB *for system with* $2n$ *equations (left) and* $n+1$ *equations (right) where* n *is the number of variables. The vertical dashed line separates the parameter with different degree of regularity (DoR). The corresponding data is described in Table 7.*

		CPU time (sec)		Memory(MB)	
n	m	M4GB	Magma	M4GB	Magma
20	40	73.28	178.41	78	361.48
21	42	222.27	384.43	117	577.34
22	44	560.11	1242.39	230	853.84
23	46	1425.68	2447.40	354	1324.16
24	48	3474.01	23951.03	696	8872.94
25	50	7647.64	59692.81	1416	19718.78
26	52	21432.34	141111.18	3049	25197.97
27	54	64944.98	314647.02	6181	39844.84

Table 8: *Performance comparison of* M4GB *and Magma for systems with* n *variables and* $m = 2n$ *equations over* $\mathbb{F}_{31}$.

		CPU time (sec)		Memory(MB)	
n	m	M4GB	Magma	M4GB	Magma
10	11	1.74	2.35	23	32
11	12	4.36	7.98	24	64
12	13	15	42.20	51	114
13	14	51	204.57	104	281
14	15	326.96	2007.68	363	1104
15	16	1951.04	12332.51	1046	3320

Table 9: *Performance comparison of* M4GB *and Magma for systems with* n *variables and* $m = n + 1$ *equations over* $\mathbb{F}_{31}$.

6 Solving MQ Challenges

Contents

In Section 2.2, we have discussed the computational hardness of the MQ problem that can be used as a basis for the security of public-key cryptography even in the presence of quantum computers. There are several constructions of public-key cryptography schemes whose security are based on the hardness of the MQ problem over finite fields such as [FPR17a, DCP+17, BPSV17]. While theoretical asymptotic complexity analysis to solve the MQ problem is important, it is also necessary to assess the concrete hardness of this computational problem.

In practice, studying the resources required to solve concrete MQ problem instances can be used to determine security parameters for cryptographic schemes that are deemed sufficient for concrete security levels. The difficulty of solving the MQ problem over a finite field depends on several factors such as order of the finite field, number of variables, number of equations, sparsity/density of the system, etc. In order to investigate the impact of these parameters to the complexity of solving the MQ problem in practice, one needs to construct concrete instances of the MQ problems with realistic parameters. By realistic parameters, we refer to parameters such that corresponding MQ problems are neither too easy nor too hard to solve in practice.

In April 2015, Yasuda et al. [YDH+15a] presented an open public challenge for the MQ problem with various parameters and increasing difficulty. One of the aims of the challenge is to evaluate the current state-of-the-art implementation of different algorithms that can solve MQ problems over finite fields. The official website of the MQ challenges can be found in the following link

`https://www.mqchallenge.org`.

In this chapter, we will describe these challenges and our efforts to solve them. Section 6.1 gives an overview of the MQ challenge, the rationale behind categorization of the challenges and how the parameters were selected. Section 6.2 describes existing state-of-the-art implementation of algorithms that managed to find solutions for some MQ challenge parameters. In Section 6.3 we present an overview of hybrid approach to solve the MQ problem over finite fields. We use the hybrid approach to solve several signature-type MQ challenges using M4GB algorithm. We will discuss it further in Section 6.4. In Section 6.5, we will give cost estimation in terms of CPU time and memory to solve larger parameters of signature-type MQ challenge with respect to M4GB.

6.1 Overview of the MQ Challenge

The MQ challenges are sets of concrete instances of the MQ problems over finite fields in various level of difficulties. The goal of the MQ challenges is to stimulate the research on algorithms to solve the MQ problems over finite fields and to analyze their applicability in practice. The challenge was initiated following similar directions of other existing open challenges for hard computational problems used as trapdoor in public-key cryptography. For instance, the RSA factoring challenge [RSA91], the Elliptic Curve Cryptography challenge (ECC) by Certicom [ECC97], and the Lattice Challenge [SVP10]. These challenges provided many publicly accessible concrete instances of problems with various parameters in increasing difficulties.

The three parameters that needs to be set to construct an instance of the MQ problem over a finite fields are: the order of the finite field q, the number of variables n, and the number of equations m. The authors of the MQ challenges selected $\mathbb{F}_2$, $\mathbb{F}_{2^8}$, and $\mathbb{F}_{31}$ as the possible base fields for the polynomials. The extension field $\mathbb{F}_{2^8}$ is constructed using the irreducible polynomial $x^8 + x^4 + x^3 + x^2 + 1 \in \mathbb{F}_2[x]$. The rationale of choosing $\mathbb{F}_{2^8}$ and $\mathbb{F}_{31}$ is because typically operations in an even-characteristic finite field are more efficient than in odd-characteristic finite fields. This difference will affect the complexity of solving larger instances of MQ problems in practice. The choice of $\mathbb{F}_2$ is considered to be a special case since there are dedicated approaches to solve those challenges, such as [FJ03, BCC$^+$10, BCC$^+$13].

The MQ challenges consider two types of challenge constructions. The first type of construction is for encryption-type parameters and the second type is for signature-type parameters. The former is characterized by an overdefined system of equations, where $m > n$ and the latter is characterized by an underdefined system of equations where $m < n$. Note that a random system of equations over $\mathbb{F}_q$ with $m > n$ has a probability approximately $\frac{1}{q^{m-n}}$ to have a solution. In order to guarantee the existence of a solution for an overdefined system, it is necessary to first randomly select a vector $s \in \mathbb{F}_q^n$. Once the coefficients of polynomials $f_1, \ldots, f_m \in \mathbb{F}_q[x_1, \ldots, x_n]$ have been generated, the constant coefficients $c_1, \ldots, c_m$ of each polynomial $f_1, \ldots, f_m$ must be adjusted to $c_i \leftarrow c_i - f_i(s)$ for all $i \in \{1, \ldots, m\}$. In this way, the overdefined system is guaranteed to have a solution, which is the vector s itself. However, for random systems with $m < n$, it is generally unnecessary to generate a solution in advance.

For MQ challenges representing public-key encryption, the authors selected parameter values with $m = 2n$ as in the case of several existing proposals such as the ABC scheme [TDTD13] and ZHFE [PBD14]. As for signature schemes, the organizer took Rainbow [DS05] as a representative of MQ-based digital signature schemes and selected parameter values with $n \approx 1.5m$.

In total there are six (6) different types of MQ challenges. The parameters of all type of challenges are described in Table 10. Type I, II, and III consist of overdefined system of equations with $2n$ equations and n variables. Type IV, V and VI consist of underdefined systems of equations with m equations and $\approx 1.5m$ variables.

The smallest challenge parameters (n and m) for each type were chosen based on estimations that it would take at least one month to solve them. For systems over $\mathbb{F}_2$, the estimation is based on the benchmark of libFES [BCC$^+$10].

Type	$\mathbb{F}_q$	Category	(n, m)	Parameter
I	$\mathbb{F}_2$	Encryption	$m = 2n$	$55 \leq n \leq 100$
II	$\mathbb{F}_{2^8}$	Encryption	$m = 2n$	$35 \leq n \leq 54$
III	$\mathbb{F}_{31}$	Encryption	$m = 2n$	$34 \leq n \leq 53$
IV	$\mathbb{F}_2$	Signature	$n \approx 1.5m$	$55 \leq m \leq 100$
V	$\mathbb{F}_{2^8}$	Signature	$n \approx 1.5m$	$16 \leq m \leq 35$
VI	$\mathbb{F}_{31}$	Signature	$n \approx 1.5m$	$16 \leq m \leq 35$

Table 10: *Six (6) different types of MQ challenges. The smallest choice of n and m were selected based on an estimation that it would take at least one month of computation to find a solution using four 6-cores 2.9GHz Intel Xeon CPU E5-4617 equipped with 15MB Intel smart cache and 1TB of RAM.*

For systems over $\mathbb{F}_{2^8}$ and $\mathbb{F}_{31}$, the estimation is based on running the Gröbner bases algorithm implemented in Magma [BCP97] v2.19-9 on four 6-cores 2.9GHz Intel Xeon CPU E5-4617 equipped with 15MB cache and 1TB of RAM. The experiments considers the direct approach to solve the system of equations for encryption-type parameters, i.e., type I, II and III. For the signature-type parameters (type IV, V, and VI) the benchmark were done on square system obtained by randomly fixing the value of $n - m$ variables. The results also estimated that the time complexity to solve encryption-type parameters and signature-type parameters should increase by a factor of 2 and 10, respectively.

6.2 Related Work

Since its public announcement, there have been various approaches to solve the MQ challenges. Some approaches were dedicated towards systems over $\mathbb{F}_2$ while others are tailored for the non binary field encryption-type or signature-type parameters. Thus we classify the approaches into three (3) different groups and we summarize them in this section.

For systems over $\mathbb{F}_2$, i.e., type I and IV, almost all approaches were equally effective to find solutions for both overdefined and underdefined systems of equations. At the time of writing there are six (6) algorithms that managed to solve type I and IV challenges: High-Performance Crossbred [BS23], exhaustive search with gray-code enumeration [BCC$^+$13], Crossbred algorithm [JV17], a variant of Crossbred algorithm implemented in GPU [NNY18], and two other algorithms that, to the best of our knowledge, have no accompanying publications (in the MQ challenge submission, the authors of these two algorithms referred to them as "improved crossbred" and "a new algorithm". We will refer to them similarly to avoid confusion). We summarize these results in Table 11. We see that the Crossbred algorithm and its variants have solved the highest parameter for type I challenge ($n = 83$) and type IV challenge ($m = 76$).

For non-binary field overdefined systems of equations, i.e., type II and III challenges, there are three (3) algorithms that managed to find solutions for some of the parameters: the XL algorithm [CCNY12], the "Modified-F_4" algorithm, and the "F_4-style" algorithm. However, we are not aware of any publications that described the last two algorithms or their implementations. The XL algorithm in [CCNY12] uses the Block Wiedemann algorithm [Cop94] to find a solution for the sparse system of linear equations defined over a finite field,

Algorithms	Type		Reference
	I (n)	IV (m)	
High-Performance Crossbred	75-77, 80, 83	60, 64, 68, 71-72, 75-76	[BS23, JV17]
Improved Crossbred†	74	68-69	-
Crossbred-GPU	55-74	67	[NNY18]
"A new algorithm"†	55-62, 67, 69	55, 60-67	-
Crossbred	67-74	-	[JV17]
Exhaustive Search	55-66	55-66	[BCC+13]

Algorithms	Resources	
High-Performance Crossbred	$n = 75, 76, 77$	1024 AMD EPYC 7J13 cores
	$n = 80$	2048 AMD EPYC 7J13 cores
	$n = 83$	3488 AMD EPYC 7J13 cores
	$m = 60$	Intel Xeon Gold 6240 (40 cores)
	$m = 64$	Intel Xeon Gold 6240 (2560 cores)
	$m = 68$	Intel Xeon Gold 6240 (5120 cores)
	$m = 71$	Intel Xeon Gold 6248 (10240 cores)
	$m = 72, 75, 76$	Intel Xeon Gold 6248 (20480 cores)
Improved Crossbred†	8x Intel i7 8700 with 32GB RAM and 10x GeForce GTX 1080Ti	
Crossbred-GPU	$n = 55, \ldots, 67$	Nvidia GTX 980 graphics card.
	$n = 68, \ldots, 74$	27 x AMD FX(tm)-8350 with 11 x Nvidia GTX 780 CUDA v7.5, 1 x Nvidia GTX 780 CUDA v8.0 and 15 x Nvidia GTX 980 CUDA v7.5.
"A new algorithm"†	Intel i9-7900X @ 3.30 GHz and a cluster of Intel Xeon @ 2.6 GHz	
Crossbred	Heterogeneous cluster of Intel Xeon @ 2.7-3.5 Ghz. The number of cores used for $n = 67, \ldots, 74$ are respectively equal to $256, 192, 512, 4096, 1760, 5320, 608$ and 448.	
Exhaustive Search	Rivyera, 128 Spartan 6 FPGAs.	

Table 11: *Summary of algorithms and resources used to solve type I and IV challenges.*

Algorithms	Type		Reference
	II (n)	III (n)	
XL	-	$34-38$	[CCNY12]
F_4-style†	$36-37$	37	-
Modified-F_4†	35	34	-

Algorithms	Resources	
XL	$n = 34, \ldots, 36$	4 x AMD Opteron 6282 SE
	$n = 37, 38$	2x AMD EPYC 7742
F_4-style†	4 x Intel(R) Xeon(R) CPU E5-4669 v4, 2.20GHz, 1TB RAM	
Modified-F_4†	Intel(R) Xeon(R) CPU E5-2620 v4, 2.10GHz with 256GB RAM	

Table 12: *Summary of algorithms and resources used to solve type II and III challenges.*

which is the case of the linearized system in XL. This algorithm currently holds the record for type III challenge with $n = 38$ while the "F_4-style" algorithm holds the record for type II challenge with $n = 37$. We summarize these results in Table 12.

For non-binary field underdefined systems of equations, i.e., type V and VI, there are six (6) algorithms that successfully found solutions for some of the parameters: M4GB [MS17], F_4 based on Hilbert-driven algorithm [ST23], Improved-F_4, MiniGB-F_4, M4GB 1.2 and Faugère's F_5 algorithm [Fau02]. Note that at the time of writing, we are not aware of any accompanying publications that describe MiniGB-F_4, Improved-F_4 and M4GB 1.2 algorithms. However a brief description of the MiniGB-F_4 can be found in the abstract of the following presentation [Che16]. It mentioned MiniGB-F_4 as a Buchberger-style Gröbner bases algorithm that maintains minimal basis throughout the computation. For the Improved-F_4 algorithm, the description given in the MQ challenge submission is that the algorithm takes advantages of the sparsity of the matrices combined with GPU-accelerated elimination steps.* The M4GB algorithms hold the record for both type V and VI respectively for the period of 5.8 years (Oct 2017-Aug 2023) and 6.3 years (Jul 2017-Oct 2023).

6.3 Overview of Hybrid Approach

One of the properties of polynomials in MPKC schemes is that they are defined over a finite field. In some cases, the order of the finite field is relatively "small". For instance, the public-key of Gui [DCP+17], one of the multivariate-based digital signature proposal for NIST post-quantum cryptography, uses polynomials over $\mathbb{F}_2$. Another NIST post-quantum proposal LUOV [BPSV17] uses polynomials over $\mathbb{F}_{2^r}$ for $r \in \{8, 48, 64, 80\}$.

The hybrid approach to solve a zero-dimensional system of multivariate polynomials over a finite field is a combination of a Gröbner basis algorithm and exhaustive search. The method was proposed by Bettale, Faugère, and Perret

† We are not aware of any publication that mentioned this approach in detail.
* `https://www.mqchallenge.org/details/details_V_20230822.html`

Algorithms	Type		Reference
	V (m)	VI (m)	
F_4 based on Hilbert-driven	-	$21-22$	[ST23]
Improved-F_4[†]	20	-	-
M4GB	$16-19$	$16-20$	[MS17]
MiniGB-F_4	$18, 19$	-	[Che16]
M4GB1.2[†]	$16-17$	-	-
F_5	16	16	[Fau02]

Algorithms	Resources	
F_4 based on Hilbert-driven	AMD EPYC 7742, 2TB RAM	
Improved-F_4	1 x AMD Ryzen Threadripper 3970X, 192GB RAM, 2 x GeForce RTX 3080Ti	
M4GB	2 x Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz and 128GB RAM	
MiniGB-F_4	$m = 18$	8 x Intel(R) Xeon(R) CPU E5620, 1 x Intel(R) Xeon(R) CPU E3-1230 v3 and 1 x Intel(R) Xeon(R) CPU E3-1245 v3.
	$m = 19$	2 x AMD opteron 6376x4 512GB RAM.
M4GB1.2	Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz, 32GB RAM	
F_5	Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz	

Table 13: *Summary of algorithms and resources used to solve type V and VI challenges.*

in [BFP09, BFP12]. It broke some proposed parameters of the TRMS signature scheme [WHL+05] and the UOV signature scheme [KPG99, BERW08]. The hybrid approach also managed to find a collision in some proposed multivariate hash functions [DY07].

The main idea of the hybrid approach is to solve many systems with less number of variables than the original ones by fixing, say, k out of the n original variables. This approach reduces the complexity of directly solving the original system of equations using the Gröbner basis algorithm at the expense of having to solve many of these subsystems. Furthermore, we can run the Gröbner basis algorithm on each of the q^k subsystems in parallel where q is the order of the finite field.

We first recall the notion of degree of regularity as the main parameter to measure the complexity of Gröbner bases algorithm.

Definition 6.1 (Degree of Regularity [BFS04])**.** *The degree of regularity of a zero-dimensional ideal* $I = \langle f_1, \ldots, f_m \rangle \subset \mathbb{F}[x_1, \ldots, x_n]$ $(m \geq n)$ *is*

$$d_{reg} = \min \left\{ d \geq 0 : \dim_{\mathbb{F}}(\{f \in I, \deg(f) = d\} \cup \{0\}) = \binom{n+d-1}{d} \right\}.$$

Remark 6.2. *Intuitively, the degree of regularity is the smallest* d *such that the number of linearly independent nonzero polynomials of degree* d *in a zero-dimensional ideal* I *is equal to the number of monomials of degree* d*.*

In order to understand the complexity of solving random system of equations, the notion "random system" is formalized as regular sequence and semi-regular sequence [Bar04]. We are mainly interested with the notion of semi-regular sequence since a random overdefined system of equations is generally assumed to be semi-regular (see for instance Hypothesis 3.3 in [BFP09] and Hypothesis 1 in [BFP12]).

Definition 6.3 (Semi-Regular Sequence [BFP09])**.** *A sequence of homogeneous polynomials* $(f_1, \ldots, f_m) \subset \mathbb{F}[x_1, \ldots, x_n]$ *of respective degrees* $d_1, \ldots, d_m$ *is semi-regular if*

- $\langle f_1, \ldots, f_m \rangle \neq \mathbb{F}[x_1, \ldots, x_n]$ *and*
- *for all* $i \in \{1, \ldots, m\}$ *and* $g \in \mathbb{F}[x_1, \ldots, x_n]$, $g \cdot f_i \in \langle f_1, \ldots, f_{i-1} \rangle$ *and* $\deg(g \cdot f_i) < d_{reg}$ *implies that* $g \in \langle f_1, \ldots, f_{i-1} \rangle$.

In [BFS04, Proposition 6], it has been shown that the degree of regularity of a semi-regular sequence can be computed explicitly.

Proposition 6.4 (Degree of Regularity of Semi-Regular Sequence [BFS04])**.** *The degree of regularity of a semi-regular sequence* $(f_1, \ldots, f_m) \in \mathbb{F}[x_1, \ldots, x_n]$ *of respective degrees* $d_1, \ldots, d_m$ *is given by the index of the first non-positive coefficient of*

$$\sum_{k \geq 0} c_k z^k = \frac{\prod_{i=1}^{m} (1 - z^{d_i})}{(1-z)^n}.$$

The complexity analysis of hybrid approach is based on the complexity of F_5 algorithm. The complexity of computing Gröbner basis of a zero-dimensional ideal with F_5 algorithm is described in the following proposition.

Proposition 6.5 (Complexity of F_5 [BFP12, BFSY05])**.** *The complexity of computing Gröbner basis of a semi-regular zero-dimensional ideal in n variables and m equations with F_5 algorithm is*

$$\mathcal{O}\left(\binom{n+d_{reg}}{d_{reg}}^{\omega}\right) \tag{6.1}$$

where d_{reg} is the degree of regularity of the system and $2 \leq \omega \leq 3$ is the linear algebra constant.[*]

Remark 6.6. *Although the number of equations m is not explicitly mentioned in Equation 6.1, note that the degree of regularity depends on m.*

When an n-variable system of equations is defined over a finite field $\mathbb{F}_q$ and we are only concerned with solutions in $\mathbb{F}_q^n$, it is possible to find all solutions by exhaustive search, which has complexity $\mathcal{O}(q^n)$. Moreover, the exhaustive search is easily parallelizable with negligible memory cost. Thus, one can implement it in massively parallel architectures such as Graphical Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs). The effectiveness of this approach has been shown in practice by solving systems of equations over $\mathbb{F}_2$ [BCC+10, BCC+13]. In a scenario where it is not feasible to perform exhaustive search, one can run a Gröbner basis algorithm with degree-reverse lexicographic ordering followed by the FGLM algorithm (if necessary) to obtain the set of solutions.

The main question then for the hybrid approach is to find the value of k such that the total expected complexity is minimal. Before obtaining the total expected complexity of the hybrid approach, we first need to understand the complexity of solving a single subsystem with m equations and $n-k$ variables. The authors of [BFP09] made the following assumption.

Assumption 6.7. *Let $\{f_1, \ldots, f_m\} \subset \mathbb{F}_q[x_1, \ldots, x_n]$ be a semi-regular system of equations with degree d. For all $0 \leq k \leq n$ and $(v_1, \ldots, v_k) \in \mathbb{F}_q^k$, we assume that*

$$\{f_1(x_1, \ldots, x_{n-k}, v_1, \ldots, v_k), \ldots, f_m(x_1, \ldots, x_{n-k}, v_1, \ldots, v_k)\}$$

is also semi-regular system of equations.

The complexity of the hybrid approach for semi-regular system of equations is stated in the following proposition.

Proposition 6.8 ([BFP12])**.** *Using Stirling's approximation, i.e. $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, the complexity of solving a semi-regular system of equations with a hybrid approach using F_5 algorithm is equal to*

$$q^k \left(\frac{1}{\sqrt{2\pi}} \cdot \frac{(n-k+d_{reg}(k))^{n-k+d_{reg}(k)+\frac{1}{2}}}{(n-k)^{n-k+\frac{1}{2}} d_{reg}(k)^{d_{reg}(k)+\frac{1}{2}}}\right)^{\omega}$$

where $2 \leq \omega \leq 3$ is the linear algebra constant and $d_{reg}(k)$ is the degree of regularity of each subsystem obtained after fixing k variables.

[*] The complexity of F_5 algorithm mentioned in [BFP09] is equal to $\mathcal{O}\left(\left(m \cdot \binom{n+d_{reg}-1}{d_{reg}}\right)^{\omega}\right)$. This was later revised in [BFP12] since Equation 6.1 is a more appropriate complexity estimation for semi-regular systems (see Assumption 6.7).

The initial result on the hybrid approach in [BFP09] suggested that one should find the optimal value of k by simply comparing the total complexities of the hybrid approach for different values of $0 \leq k \leq n$. This approach is described in Algorithm 6.1.

Input: q - order of the finite field
Input: n - number of variables of the system
Input: m - number of equations of the system
Input: $d_1, \ldots, d_m$ - the degree of each polynomial
Result: k - the best theoretical trade-off for hybrid approach
1 $A \leftarrow [\,]$
2 **for** $k \leftarrow 0$ **to** n **do**
3 $\quad d_k \leftarrow$ index of the first non-positive coefficient in the series $\frac{\prod_{i=1}^{m}(1-z^{d_i})}{(1-z)^{n-k}}$
4 $\quad A[k] \leftarrow q^k \left(\frac{1}{\sqrt{2\pi}} \cdot \frac{(n-k+d_k)^{n-k+d_k+\frac{1}{2}}}{(n-k)^{n-k+\frac{1}{2}} \cdot d_k^{d_k+\frac{1}{2}}} \right)^{\omega}$
5 **return** k such that $A[k]$ is minimum.

Algorithm 6.1: An algorithm to find the optimal value of k in hybrid approach.

An improved analysis of hybrid approach in [BFP12] provided an explicit formula for the best trade-off k to solve a system of quadratic equations, that is

$$\left\lceil n \cdot \frac{10.86 \cdot \omega^2}{(4.16 \cdot \log_2 q - 3.14 \cdot \omega)^2} \right\rceil \tag{6.2}$$

(see [BFP12, Proposition 3.3]). Using Equation 6.2, the complexity of the hybrid approach for a random quadratic system of equations over $\mathbb{F}_q$ is asymptotically equivalent to

$$2^{n\omega(1.38-0.63\omega \log_2(q)^{-1})} \tag{6.3}$$

when $n \to \infty, q \to \infty$ and $\log_2(q) \ll n$ (see [BFP12, Theorem 3.1]).

We would like to remark that a similar approach was also proposed for the XL algorithm in [CKPS00] called the FXL (Fixed XL) algorithm. The paper stated that the complexity of solving a system of equations using the XL algorithm can be reduced by guessing a small number of variables. The asymptotic complexity of the FXL was further described in [YCC04]. Recall that the XL algorithm is a redundant variant of the F_4 algorithm, thus it is considered to be less efficient than both the F_4 and the F_5 algorithm. Furthermore, the asymptotic analysis of the FXL algorithm in [YCC04] did not yield a method to determine the optimal number of guessed variables.

6.4 Solving Type V and VI of the MQ Challenge

In this section we will discuss how the M4GB algorithm was able to find solutions of some of the MQ challenges. The M4GB algorithm managed to solve signature-type parameters (underdefined systems) with coefficients in $\mathbb{F}_{31}$ and $\mathbb{F}_{2^8}$, i.e., type V and type VI of MQ challenge. We used the following two machines to break these challenges.

A) A desktop machine equipped with Intel(R) Core(TM) i7-2600K CPU @ 3.40 GHz with four (4) processor cores and 16GB RAM

B) A NUMA (Non-Uniform Memory Access) machine with two nodes. Each node is equipped with Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz with ten (10) processor cores and 128GB RAM.

We refer to the above machines as machine A and B respectively. The machine A was used to find solutions for the smallest parameter of type V and type VI, which consist of 24 variables and 16 equations defined over $\mathbb{F}_{2^8}$ and $\mathbb{F}_{31}$ respectively. Larger parameters were solved by running M4GB on machine B. A summary of challenges solved by M4GB is shown in Table 14. The solutions found are provided in Appendix A. It is expected for a random underdefined system of equations to have multiple solutions. However, it is sufficient to find a single solution in the base field in order to break a given parameter.

Our strategy to solve type V and VI MQ challenges uses the hybrid approach. We first fix the value of $n - m$ variables in order to yield a square system with equal number of variables and equations. This will be followed by the hybrid approach iterating over all possible values of another k number of variables. In most cases, we selected $k = 1$ except for the largest parameter of type VI that we solved, where we selected $k = 2$. A summary of this strategy is given in Algorithm 6.2.

It is possible that the square system F' in Algorithm 6.2 has no solution. For a system over a prime field, it is known that a random instance of a square system with quadratic polynomials has no solution with probability $1/e$ asymptotically in n [FB09, Corollary 3.1].* If F' eventually had no solution, the Algorithm 6.2 would run another iteration to generate a new F' by choosing a new $(v_1, \ldots, v_t) \in \mathbb{F}_q^t$. The benefit of this approach is that the computation of Gröbner bases from line 7 up to line 19 is trivially parallelizable, i.e., we can compute Gröbner basis for each F'' independently. Moreover, we can also estimate the average complexity and worst-case complexity in practice to obtain a solution for the whole system by computing a Gröbner basis for one of the subsystems F''.

For instance, solving one subsystem of type VI with 15 variables and 16 equations took 1 hour and 10 minutes wall-clock time. An average total CPU-time estimate to solve type VI parameter with $m = 16$ would thus take 18.7 hours, assuming that the generated square system has a solution. The implementation of M4GB uses 837MB of memory to solve each subsystem, which allowed simultaneous computation on 8 subsytems within the available 16GB of memory in machine A. We broke both type V and VI challenges for $m = 16$ on the quadcore desktop machine A in 9.3 hours and 1.2 hours wall-clock time after covering 29% and 22.6% of the search space respectively. Note that this is significantly faster than the original expected one month estimate on a Xeon system using Magma by the authors of the MQ challenges.

For larger parameters for type V and VI, we used machine B. We ran 10 simultaneous Gröbner bases computations using M4GB by forcing each process to one NUMA node using the `numactl` program.

* More generally, given a multivariate polynomial system with $n+\alpha$ random equations of degree $d \geq 2$ in n variables over $\mathbb{F}_p$, with a prime p. The probability that the system has no solution is $e^{-p^{-\alpha}}$ asymptotically in n. For the proof, see [FB09, Section 3]

Input: $F = \{f_1, \ldots, f_m, x_1^q - x_1, \ldots, x_n^q - x_n\} \subset \mathbb{F}_q[x_1, \ldots, x_n]$ with $n > m$
Input: $k \in \mathbb{Z}_{>0}$
Output: A solution $(s_1, \ldots, s_n) \in \mathbb{F}_q^n$ of F (assume it exists)

$G \leftarrow \{\}$
solution_found $\leftarrow$ False
$t \leftarrow n - m$
while solution_found = False **do**
  Choose randomly $(v_1, \ldots, v_t) \in \mathbb{F}_q^t$
  $F' \leftarrow \{f(x_1, \ldots, x_m, v_1, \ldots, v_t) : f \in F\}$
  for $(w_1, \ldots, w_k) \in \mathbb{F}_q^k$ **do**
    $F'' \leftarrow \{f'(x_1, \ldots, x_{m-k}, w_1, \ldots, w_k) : f' \in F'\}$
    $G \leftarrow$ GröbnerBasis(F'') ▷ Compute the reduced Gröbner bases
    if $G \neq \{1\}$ **then**
      while $\exists g \in G : g - \mathsf{LT}(g) \notin \mathbb{F}_q$ **do**
        Let x be a variable in g
        for $c \in \mathbb{F}_q$ **do**
          $G' \leftarrow$ GröbnerBasis$(G \cup \{x + c\})$
          if $G' \neq \{1\}$ **then**
            $G \leftarrow G'$
            break
      solution_found $\leftarrow$ True
      break
G `is of the form` $\{x_1 + c_1, \ldots, x_{m-k} + c_{m-k}\}$ `where` $c_i \in \mathbb{F}_q$.
$(s_1, \ldots, s_n) \leftarrow (-c_1, \ldots, -c_{m-k}, w_1, \ldots, w_k, v_1, \ldots, v_t)$
return $(s_1, \ldots, s_n)$

Algorithm 6.2: A strategy based on hybrid approach to find a solution for an underdefined system of equations over a finite field

For type VI with $m = 19$, we modified our strategy to compute simultaneous Gröbner bases on subsystems with $n = 18$ variables. We first observed that in the final stage, the M4GB algorithm takes a significant amount of time using a single thread. Thus, instead of waiting for the whole process to finish, one can start the Gröbner basis computation for the next subsystem as soon as this last stage of the previous computation begins. Of course this requires that the available memory is sufficiently large for these concurrently running Gröbner basis computations. In this way, all processors are fully occupied and it significantly reduces the time to obtain a solution.

Type	(n, m)	Machine Used	# Node	k	Time (sec)	Memory	Duration	Search Space Covered
V*	$(24, 16)$	A	1	1	1844	1.11GB	$\approx$ 9.3 hours	29%
V	$(25, 17)$	B	1	1	24125	3.24GB	$\approx$ 46.33 hours	46.5%
V	$(27, 18)$	B	2	1	94266	11.75GB	$\approx$ 10.9 days	95.7%
V	$(28, 19)$	B	2	1	478194	45.05GB	$\approx$ 69.75 days	71.5%
VI	$(24, 16)$	A	1	1	4273	836.9MB	$\approx$ 1.2 hours	22.6%
VI	$(25, 17)$	B	1	1	30955	3.25GB	$\approx$ 9.87 hours	64.5%
VI	$(27, 18)$	B	1	1	194474	11.9GB	$\approx$ 31.48 hours	12.9%
VI	$(28, 19)$	B	2	1	474008	47.1GB	$\approx$ 7.61 days	54.8%
VI	$(30, 20)$	B	2	2	139088	12.67GB	$\approx$ 11.32 days	21.5%

Table 14: *Summary of MQ challenge parameters solved using M4GB*

* The CPU time and memory usage for this parameter were run on machine B due to the loss of information.

6.5 Cost Estimation for Other Parameters

In this section we provide a cost estimation to find solution for unsolved instances of type V and VI of MQ challenge with hybrid approach using the M4GB algorithm. We focus on estimating the total CPU time and memory usage of the implementation described in Section 5.6.

6.5.1 Approach

Recall that solving an underdefined system of equations with n variables and m equations, where $n > m$, is equivalent to solving a system with m variables and m equations obtained after fixing the value of $n - m$ variables from the original system. To simplify our methodology, we only estimate the cost of hybrid approach on systems with equal number of variables and equations and we assume that such system obtained from type V or VI of MQ challenges has a solution. We also assume that the CPU time and memory usage of M4GB implementation in Section 5.6 grow exponentially in terms of the number of variables n.

$m = n$		16	17	18	19	20	21	22	23	24	25
k	$\mathbb{F}_{2^8}$	2	1	2	1	2	1	2	2	2	2
	$\mathbb{F}_{31}$	4	5	5	3	5	4	6	5	4	6

$m = n$		26	27	28	29	30	31	32	33	34	35
k	$\mathbb{F}_{2^8}$	**3**	2	**3**	2	**3**	**3**	**3**	**3**	2	**3**
	$\mathbb{F}_{31}$	5	7	6	5	7	6	**8**	7	6	**8**

Table 15: *Optimal number of fixed variables k for F_5 algorithm applied to type V and type VI parameters of MQ challenge using Algorithm 6.1 assuming $\omega = 2.4$. The numbers highlighted in bold are the maximum k for the given finite field.*

Note that since type V and VI challenges are parametrized by the number of equations m and we restrict our approach on systems with $n = m$ variables, the results in this section are described in terms of m in order to give a unified picture of the estimation.

For a given number of equations/variables m and an order of a finite field q, the first step of hybrid approach is to find the optimal value of the number of fixed variables k. Here by optimal we mean the one that minimizes the total CPU time.* In order to find the optimal value k in practice, we shall examine the cost of running M4GB algorithm on subsystems generated after fixing k variables.

For each q and m, we run the M4GB algorithm for systems with $m - 1, m - 2, \ldots, m - k_{\max}$ variables where $k_{\max}$ is the maximum number of fixed variables. We select the value $k_{\max}$ by finding the maximum number of fixed variables for a given q using the theoretical result described in Algorithm 6.1

* Note that it is also possible to define the optimal value k in terms of other costs such as memory usage, energy, monetary cost, etc. We leave the study on those costs for future work.

for $n = m = 16, \ldots, 35$ and $d_1 = \ldots = d_m = 2$. We obtain $k_{\max} = 3$ for $q = 256$ and $k_{\max} = 8$ for $q = 31$. The complete result is given in Table 15.

After obtaining $k_{\max}$ for both $q = 31$ and $q = 256$, we collected the CPU time and memory usage of M4GB algorithm on subsystems of type V and VI starting from $m = 16$ up to the number of equations that are feasible to solve using a single node of machine B. The full result is described in Appendix B. The total CPU time and total memory usage are the product of the CPU time and memory usage by q^k. To simplify the estimation, we took the base-2 logarithm of the total CPU time as well as total memory usage and use the *least square method* to predict the cost for the unsolved parameters.

6.5.2 Result

For type V, our estimation suggests that guessing one variable from the square system is the optimal strategy for all the remaining unsolved challenge parameters, i.e., $m = 20, \ldots, 35$. This can be seen from the growth of the CPU hours for type V challenge in Figure 11 and the detail description in Table 16. For the next unsolved parameter, i.e., $m = 20$, we estimate that it would cost approximately 402045 CPU hours $\approx$ 45.9 CPU years and 52.61 TB of memory.

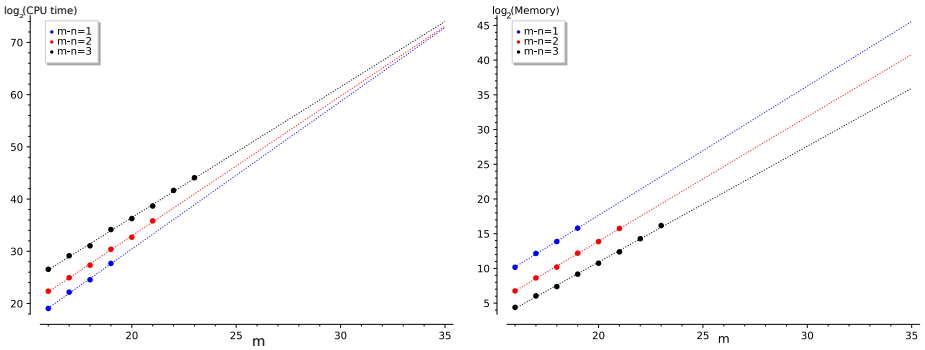


Figure 11: *Estimated growth of total CPU time (left) and per-subsystem memory usage (MB) (right) of hybrid approach for type V MQ challenge using the* M4GB *algorithm.*

For type VI challenge, the optimal number of variables to fix differs for different values of m. Our estimation suggests to fix 3 variables from the generated square system to solve the next unsolved parameter, which is $m = 21$. The cost to solve the next unsolved parameter, i.e., $m = 21$, would approximately takes 217452 CPU hours $\approx$ 24.82 CPU years and 159.17 TB. However, this is not the case for $m = 22, 23$ where guessing 2 variables from the generated square system is the optimal strategy to minimize the total CPU time. For $m \geq 24$, our estimation suggests to fix 3 variables except for the largest challenge parameter $m = 35$ where it suggests to fix 5 variables. The results are described in Table 16 and Figure 12.

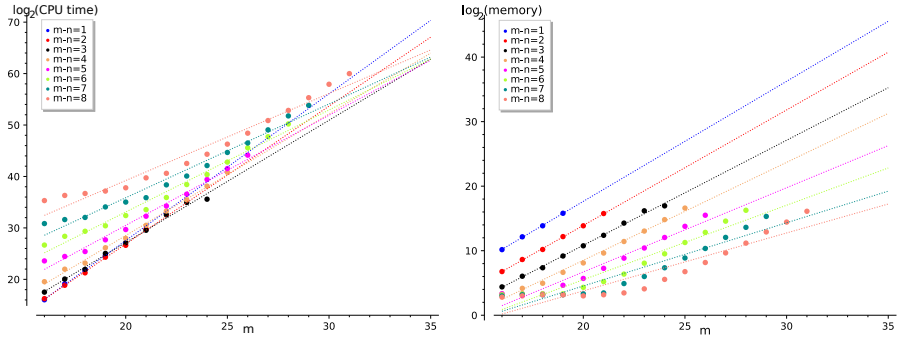


Figure 12: *Estimated growth of total CPU time (left) and per-subsystem memory usage (MB) (right) of hybrid approach for type VI MQ challenge using the* M4GB *algorithm.*

m	$m-n$		≈ CPU hours		≈ Memory (GB)	
	V	VI	V	VI	V	VI
20	1	-	402045	-	210	-
21	1	3	$2.85 \cdot 10^6$	217452	764	5.47
22	1	2	$2.02 \cdot 10^7$	$1.39 \cdot 10^6$	2776	189
23	1	2	$1.44 \cdot 10^8$	$8.94 \cdot 10^6$	10087	654
24	1	3	$1.02 \cdot 10^9$	$1.43 \cdot 10^7$	36643	130
25	1	3	$7.23 \cdot 10^9$	$1.57 \cdot 10^8$	133110	528
26	1	3	$5.13 \cdot 10^{10}$	$8.15 \cdot 10^8$	483536	1631
27	1	3	$3.64 \cdot 10^{11}$	$4.24 \cdot 10^9$	$1.76 \cdot 10^6$	5042
28	1	3	$2.58 \cdot 10^{12}$	$2.20 \cdot 10^{10}$	$6.40 \cdot 10^6$	15584
29	1	3	$1.83 \cdot 10^{13}$	$1.15 \cdot 10^{11}$	$2.32 \cdot 10^7$	$4.81 \cdot 10^4$
30	1	3	$1.30 \cdot 10^{14}$	$5.95 \cdot 10^{11}$	$8.44 \cdot 10^7$	$1.49 \cdot 10^5$
31	1	3	$9.23 \cdot 10^{14}$	$3.10 \cdot 10^{12}$	$3.06 \cdot 10^8$	$4.61 \cdot 10^5$
32	1	3	$6.55 \cdot 10^{15}$	$1.61 \cdot 10^{13}$	$1.11 \cdot 10^9$	$1.42 \cdot 10^6$
33	1	3	$4.64 \cdot 10^{16}$	$8.37 \cdot 10^{13}$	$4.04 \cdot 10^9$	$4.40 \cdot 10^6$
34	1	3	$3.30 \cdot 10^{17}$	$4.35 \cdot 10^{14}$	$1.47 \cdot 10^{10}$	$1.36 \cdot 10^7$
35	1	5	$2.34 \cdot 10^{18}$	$1.95 \cdot 10^{15}$	$5.32 \cdot 10^{10}$	$8.37 \cdot 10^4$

Table 16: *Estimated cost to solve larger parameters of type V (top) and VI (bottom) using our implementation of* M4GB *algorithm running on Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30 GHz. Memory usage is for a single subsystem and the total system memory required depends on the number of subsystems being solved simultaneously.*

7 Solving Binary Puzzles using Gröbner Bases Algorithms

Contents

This chapter presents an application of Gröbner bases to solve binary puzzles.* These are Sudoku-like puzzles where the entries are taken from the set $\{0, 1\}$. In this chapter we study the reduction of the problem of solving a binary puzzle into a problem of solving a system of multivariate polynomial equations over three different rings. The first approach reduces the problem into the problem of solving system of polynomials over $\mathbb{F}_2$. This is a natural approach since we can view each entry of a binary puzzle as an element of $\mathbb{F}_2$. The second approach reduces the problem into a system of polynomials over $\mathbb{Q}$ or $\mathbb{Z}$. The later approach decreases the degree of some polynomials into linear ones at the expense of defining the polynomials over different rings. The solutions for these system of equations correspond to the correct configurations for the binary puzzle (provided that a solution exists). Thus, we can run Gröbner bases algorithms on these systems of equations to obtain a solution for the binary puzzle. We compare the performance of Gröbner bases algorithm in Magma and SageMath to solve systems of equations from binary puzzle of various sizes over $\mathbb{F}_2$, $\mathbb{Q}$, and $\mathbb{Z}$.† The result of this chapter is an extension of the published work in collaboration with Utomo [UM17].

We first give an overview of binary puzzles in Section 7.1. This is followed by short description of Constraint Satisfaction Problem (CSP) in Section 7.2. In Section 7.3 we give an overview of prior related works. We then describe necessary notations and preliminaries in Section 7.4. Most of the contents of Section 7.4 is related to the theory of cryptographic Boolean functions, which can be found in [Car10]. The reduction of solving a binary puzzle into solving a system of polynomials over $\mathbb{F}_2$ is described in Section 7.5. This is followed by another reduction into a system polynomials over $\mathbb{Q}$ and $\mathbb{Z}$ in Section 7.6. Finally we conclude this chapter by comparing the performance and memory usage of existing Gröbner bases implementations on various sizes of binary puzzles in Section 7.7.

* also known as Takuzu/Binero/Bineiro/Zernero/Binary Sudoku.

† We could not use M4GB in this case since it lacks optimization for polynomials over $\mathbb{F}_2$ and it does not support Gröbner bases computation over $\mathbb{Z}$ and $\mathbb{Q}$.

7.1 Overview of Binary Puzzles

Binary puzzles are Sudoku-like puzzles in which the value in each cell is taken from the set $\{0, 1\}$ instead of $\{0, 1, \ldots, 9\}$. A binary puzzle is represented by an $n \times n$ square matrix where $n \geq 4$. For some entries explicit values are initially given, while the remaining entries are left undetermined. The goal of a player is to fill every entry in the puzzle while observing the following constraints:

1. (Three-value constraint) There exist no three consecutive entries, both vertically and horizontally, which are filled with the same value,
2. (Balancedness constraint) every row and column is *balanced*, i.e., in each row and each column the number of ones and zeros must be equal,
3. (Distinctness constraint) there exist no two equal rows. Similarly, there exist no two columns that are equal.

An example of initial configuration of a binary puzzle and its corresponding solution is available in Figure 13. The gray-colored entries indicate that their values are undetermined. The black and white cells represent zeroes and ones respectively (or vice versa).

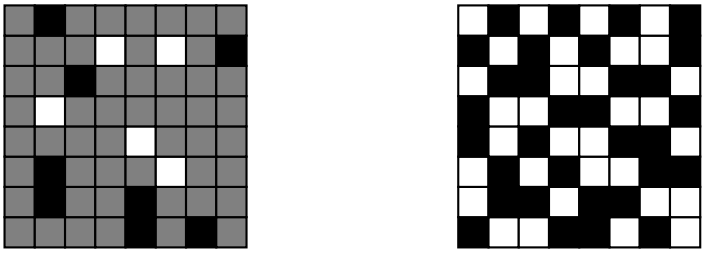


Figure 13: *An example of initial configuration of an 8×8 binary puzzle (left) and its corresponding solution (right). Note that multiple solutions may exist depending on the size and the initial configuration.*

7.2 Constraint Satisfaction Problem

The problem of solving a binary puzzle can be seen as a *constraint satisfaction problem* (CSP) [RN10, Chapter 6]. A CSP consists of the following three components

1. A set of variables $X = \{X_1, \ldots, X_n\}$,
2. A set of domains $D = \{D_1, \ldots, D_n\}$ where each domain D_i is the set of permitted values for the corresponding variable X_i,
3. A set of constraints $C = \{C_1, \ldots, C_k\}$ where each constraint C_i consists of a pair $(S, \texttt{rel})$ where S is an element of the power set of X which tells the variables that participate in the constraint and `rel` is a relation that defines the values that the variables in S can take on.

We speak about the relation `rel` in a loose fashion since there are multiple ways to describe it. A relation can be described as an explicit list of all tuples of

values that satisfy the constraint. For instance, a constraint that is satisfiable when two variables X_1, X_2 are not equal and both have the domain $\{0,1\}$ can be described as $(\{X_1, X_2\}, \{(0,1),(1,0)\})$. Another approach to describe the relation `rel` is by using a Boolean formula. For instance, the constraint in the previous example can be described as $(\{X_1, X_2\}, X_1 \neq X_2)$.

Described as a CSP, the problem of solving an $n \times n$ binary puzzle consists of n^2 variables. For $1 \leq i, j \leq n$, we denote by $x_{i,j}$ the variable that represents the entry at row i and column j. The domain $D_{i,j}$ of each variable $x_{i,j}$ is equal to $D_{i,j} = \{0,1\}$. For the three-value constraint, the number of CSP constraints for each row or column is equal to $n-2$. In total, there are $2n(n-2)$ constraints to describe it, each involving three variables. The number of constraints needed to describe the balancedness constraint is $2n$ where each of them involves n variables. Lastly, the number of constraints to describe the distinctness constraint is $2 \cdot \binom{n}{2} = n(n-1)$ where each of them involves $2n$ variables. In total, there are $3n(n-1)$ constraints to describe an $n \times n$ binary puzzle as a CSP.

7.3 Related Work

The complexity of solving binary puzzles was studied in [dB12]. Utomo and Pellikaan studied binary puzzle as a constrained array and analyzed its properties from erasure correcting point of view [UP15]. One direction to solve binary puzzles was also proposed by the same author in [UP15, UP17] by expressing the constraints as Boolean satisfiability problem in conjunctive normal form. An off-the-shelf SAT solvers (e.g., CryptoMiniSAT [Soo16]) was employed to find a solution for a binary puzzle. Utomo also proposed another approach that models the problem of solving binary puzzles as an instance of satisfiability modulo theory [Uto17, Mon16]. A physical zero-knowledge proof to verify the solution of a binary puzzle was developed in [BDDL16, Section 3].

The use of a system of multivariate polynomials to represent the constraints in a finite-domain CSP was studied in [JJGvD13]. One primary advantage in using multivariate polynomials to represent constraints in a CSP is that it allows different forms of constraint to be treated in a uniform way. The polynomials in [JJGvD13] are treated strictly as elements of $\mathbb{C}[x_1, \ldots, x_n]$. In order to restrict the domain of each variable to a finite set, say $D \subset \mathbb{C}$, the polynomials of the form $\prod_{j \in D}(x - j)$ are included in the system of equation. The paper also demonstrated how different types of constraints (constant, disjunctive, inequality, etc) can be expressed as multivariate polynomials. Finally, the authors suggested to use Gröbner bases algorithms to solve a system of equations that represents a CSP.

7.4 Notations and Preliminaries

We denote by $\mathrm{wt}(u)$ the *Hamming weight* of $u \in \{0,1\}^n$, that is the number of nonzero components of u. The vector $v = (v_1, \ldots, v_n) \in \mathbb{F}_2^n$ is said to be covered by $w = (w_1, \ldots, w_n) \in \mathbb{F}_2^n$ (or w covers v), denoted by $v \preceq w$, if $v_i \leq w_i$ for all $i \in \{1, \ldots, n\}$. For nonnegative integers $p = \sum_{i=0}^{n} p_i 2^i, q = \sum_{i=0}^{n} q_i 2^i$ we say that $p \preceq q$ if $(p_0, p_1, \ldots, p_n) \preceq (q_0, q_1, \ldots, q_n)$ where $n = \lceil \log_2(\max(p,q)) \rceil$ and $p_i, q_i \in \{0,1\}$.

An n-variable Boolean function f is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The n-variable *pseudo Boolean function* corresponding to f is the $\mathbb{Z}$-valued function φ_f on $\mathbb{F}_2^n$. By canonical ring embedding from $\mathbb{Z}$ to $\mathbb{Q}$, φ_f can also be seen as $\mathbb{Q}$-valued function. The Boolean function f can be represented as an n-variable polynomial in the ring $\mathbb{F}_2[x_1, \ldots, x_n]$

$$f = \sum_{\substack{u \in \{0,1\}^n \\ u=(u_1,\ldots,u_n)}} a_u x_1^{u_1} \cdots x_n^{u_n} \tag{7.1}$$

where $a_u \in \mathbb{F}_2$. The representation in (7.1) is called the *algebraic normal form* (ANF) of f. The coefficients a_u is computed using the following proposition.

Proposition 7.1. *Let $\sum_{u\in\{0,1\}^n} a_u x_1^{u_1} \cdots x_n^{u_n}$ be the ANF of an n-variable Boolean function f. For all $u \in \{0,1\}^n$, we have*

$$a_u = \sum_{\substack{x \in \mathbb{F}_2^n \\ x \preceq u}} f(x).$$

Proof. See [Car10, Proposition 1]. ■

Similarly, one can obtain the polynomial representation of a Boolean function f in $\mathbb{Z}[x_1, \ldots, x_n]$. We refer to such polynomial as the *numerical normal form* (NNF) of f.

Proposition 7.2. *Let $\sum_{u\in\{0,1\}^n} a_u x_1^{u_1} \cdots x_n^{u_n}$ be the NNF of an n-variable Boolean function f where $a_u \in \mathbb{Z}$. For all $u \in \{0,1\}^n$ we have*

$$a_u = (-1)^{\mathrm{wt}(u)} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \preceq u}} (-1)^{\mathrm{wt}(x)} \varphi_f(x).$$

Proof. See [Car10, Proposition 4]. ■

Remark 7.3. *Note that the ANF of a Boolean function can be computed from its NNF by reducing the coefficients modulo* 2.

Definition 7.4. *An n-variable Boolean function f is called symmetric if its output is invariant under any permutation of its input bits, that is*

$$f(x_1, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

for all permutation $\pi : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$.

From the definition above, the output of a symmetric Boolean function depends only on the Hamming weight of its input. The following notion defines an efficient representation for a symmetric Boolean function.

Definition 7.5. *For a symmetric n-variable Boolean function f, we associate a function $v_f : \{0, \ldots, n\} \mapsto \mathbb{F}_2$ such that $f(x) = v_f(\mathrm{wt}(x))$ for all $x \in \mathbb{F}_2^n$. The sequence $v(f) = (v_f(0), \ldots, v_f(n))$ is called the simplified value vector of f.*

Definition 7.6 (Elementary Symmetric Polynomials)**.** *The i-th elementary symmetric polynomials $\sigma_{i,n}$ in n-variables $x_1, \ldots, x_n$ is defined as*

$$\sigma_{i,n}(x_1, \ldots, x_n) = \sum_{1 \leq j_1 < j_2 < \cdots < j_i \leq n} x_{j_1} \cdots x_{j_i}.$$

Proposition 7.7. *An n-variable Boolean function f is symmetric if and only if its algebraic normal form can be written as*

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{n} \lambda_f(i) \cdot \sigma_{i,n}(x_1, \ldots, x_n), \quad \lambda_f(i) \in \mathbb{F}_2.$$

Proof. See [CV05, Proposition 1]. ■

Definition 7.8. *The coefficients of the ANF of a symmetric Boolean function f can be represented by the sequence of $(n+1)$-bit $\lambda(f) = (\lambda_f(0), \ldots, \lambda_f(n))$ called the simplified ANF vector of f.*

Proposition 7.9. *Let f be an n-variable symmetric Boolean function. The simplified ANF vector $\lambda(f)$ is related with its simplified value vector $v(f)$ by*

$$\lambda_f(i) = \sum_{k \preceq i} v_f(k).$$

Proof. See [CV05, Proposition 2]. ■

7.5 Polynomial Equations over $\mathbb{F}_2$

In this work, we investigate the use of systems of polynomials to specify constraints of binary puzzles. Since the domain of each variable is $\{0, 1\}$, the first natural approach is to treat this set as the binary field $\mathbb{F}_2$. This allows each constraint to be viewed as an k-variable Boolean function, where k is the number of variables involved in the constraint. For instance, the three-value constraint is a 3-variable Boolean function, whereas the balancedness constraint is an n-variable Boolean function. We say that a Boolean function allows a specific value assignment for its variables if and only if the Boolean function output equals 0. This is in line with the notion of a solution to a system of polynomials.

Recall that an n-variable Boolean function can be represented as a multivariate polynomial in (7.1) over $\mathbb{F}_2$. We use this observation and Proposition 7.1 to derive the polynomial that represents the three-value constraint.

Theorem 7.10. *Let x, y, z be variables that represent three adjacent cells (horizontally or vertically) of an $n \times n$ binary puzzle. The three-value constraint of an $n \times n$ binary puzzle is represented by the following polynomial over $\mathbb{F}_2$*

$$xy + xz + x + yz + y + z + 1. \tag{7.2}$$

Proof. Let f_1 be a Boolean function that represent the three-value constraint of binary puzzle. By definition f_1 is defined as

$$f_1(x, y, z) = \begin{cases} 1 & \text{if } (x, y, z) = (0, 0, 0) \textbf{ or } (x, y, z) = (1, 1, 1) \\ 0 & \text{otherwise.} \end{cases} \tag{7.3}$$

By Proposition 7.1, the coefficients a_u of the ANF of f_1 can be computed as:

$$a_u = \begin{cases} 0 & \text{if } u = (1,1,1) \\ 1 & \text{otherwise.} \end{cases}$$

Therefore $f_1(x,y,z) = xy + xz + x + yz + y + z + 1$. ■

We also use a similar approach to derive the polynomial representation for the balancedness constraint. We first recall the following result.

Lemma 7.11 (Lucas' Theorem [Luc78])**.** *Let n, m be nonnegative integers. A binomial coefficient $\binom{m}{n}$ is divisible by a prime p if and only if at least one of the digits in p-ary expansion of n is greater than the corresponding p-ary digit of m.*

Corollary 7.12. *Let n, m be nonnegative integers. A binomial coefficient $\binom{m}{n}$ is not divisible by 2 if and only if $n \preceq m$.*

Theorem 7.13. *Let $x_1, \ldots, x_n$ be variables that represent the n cells in a single row/column of an $n \times n$ binary puzzle. The balancedness constraint of binary puzzle is represented by the following polynomial over $\mathbb{F}_2$*

$$1 + \sum_{\substack{u \in \mathbb{F}_2^n \setminus \{0\} \\ n/2 \preceq \mathrm{wt}(u)}} x_1^{u_1} \cdots x_n^{u_n}. \tag{7.4}$$

Proof. Let $f_2 : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be a Boolean function that represents the balancedness constraint. The function f_2 is defined as

$$f_2(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \mathrm{wt}((x_1, \ldots, x_n)) = n/2 \\ 1 & \text{otherwise.} \end{cases}$$

By Proposition 7.1, the constant coefficient a_0 in the ANF of f_2 is equal to $a_0 = f_2(0) = 1$. For the coefficients a_u such that $u \in \mathbb{F}_2^n \setminus \{0\}$, we will handle the cases for $\mathrm{wt}(u) < n/2$ and $\mathrm{wt}(u) \geq n/2$ separately.

For all nonzero $u \in \mathbb{F}_2^n$ such that $\mathrm{wt}(u) < n/2$ we define the set $S_u = \{w \in \mathbb{F}_2^n : w \preceq u\}$. By Proposition 7.1 , we have $a_u = 0$ since $f_2(w) = 1$ for all $w \in S_u$ and the cardinality of S_u is even.

We will now determine the coefficients a_u for all nonzero $u \in \mathbb{F}_2^n$ such that $\mathrm{wt}(u) \geq n/2$. Let $\mathrm{Supp}(u) = \{i : u_i \neq 0\}$ denotes the support of u. The number of $w \in \mathbb{F}_2^n$ such that $\mathrm{wt}(w) = n/2$ and $w \preceq u$ is equal to $\binom{|\mathrm{Supp}(u)|}{n/2} = \binom{\mathrm{wt}(u)}{n/2}$. Therefore, $a_u = 1$ if and only if $\binom{\mathrm{wt}(u)}{n/2}$ is odd, i.e., if and only if $n/2 \preceq \mathrm{wt}(u)$ by Corollary 7.12. Therefore,

$$f_2(x_1, \ldots, x_n) = 1 + \sum_{\substack{u \in \mathbb{F}_2^n \setminus \{0\} \\ n/2 \preceq \mathrm{wt}(u)}} x_1^{u_1} \cdots x_n^{u_n}.$$

■

Our next aim is to simplify the polynomial representation for the balancedness constraint by taking into account some of the properties of the corresponding Boolean function. Note that the Boolean function that represents the balancedness constraint of binary puzzle is symmetric. Following this observation, we derive the following result.

Theorem 7.14. *Let $x_1, \ldots, x_n$ be variables that represent n entries in a single row/column of an $n \times n$ binary puzzle. The balancedness constraint of binary puzzle is represented by the following polynomial over $\mathbb{F}_2$*

$$1 + \sum_{\substack{i=1 \\ n/2 \preceq i}}^{n} \sigma_{i,n}(x_1, \ldots, x_n).$$

Proof. Let $f_2 : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be the Boolean function that represents the balancedness constraint of binary puzzle. Since f_2 is a symmetric function, by Proposition 7.7 the algebraic normal form of f_2 can be written as

$$f_2(x_1, \ldots, x_n) = \sum_{i=0}^{n} \lambda_{f_2}(i) \cdot \sigma_{i,n}(x_1, \ldots, x_n), \quad \lambda_{f_2}(i) \in \mathbb{F}_2.$$

The coefficients $\lambda_{f_2}(i)$ can be computed from the simplified value vector v_{f_2} of f_2. The function v_{f_2} is defined as

$$v_{f_2}(i) = \begin{cases} 0 & \text{if } i = n/2, \\ 1 & \text{otherwise.} \end{cases}$$

Recall that by Proposition 7.9, $\lambda_{f_2}(i) = \sum_{k \preceq i} v_{f_2}(k)$. For a nonzero i, the cardinality of the set $\{k \mid k \preceq i, \forall 0 \leq k \leq n\}$ is even. Because $v_{f_2}(i) = 0$ only for $i = n/2$, then for all nonzero i, we have

$$\lambda_{f_2}(i) = \begin{cases} 0 & n/2 \npreceq i \\ 1 & n/2 \preceq i. \end{cases}$$

For $i = 0$, by definition $\lambda_{f_2}(0) = 1$. Therefore

$$f_2(x_1, \ldots, x_n) = 1 + \sum_{\substack{i=1 \\ n/2 \preceq i}}^{n} \sigma_{i,n}(x_1, \ldots, x_n).$$

■

Finally, the following theorem gives a polynomial representation for the distinctness constraint of binary puzzle. The proof is straightforward from the definition of the distinctness constraint.

Theorem 7.15. *Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be variables that represent two distinct rows (or columns) of an $n \times n$ binary puzzle. The polynomial representation over $\mathbb{F}_2$ for the distinctness constraint of the binary puzzle is given by*

$$\prod_{i=1}^{n} (x_i + y_i + 1). \tag{7.5}$$

For $1 \leq i, j \leq n$, let $x_{i,j}$ be variable that represent the value at row i and column j of an $n \times n$ binary puzzle. The system of equations over $\mathbb{F}_2$ that represent an $n \times n$ binary puzzle is

$$
\begin{array}{ll}
x_{i,j+1}x_{i,j+2} + x_{i,j}x_{i,j+2} + x_{i,j+2} & \\
\qquad +x_{i,j}x_{i,j+1} + x_{i,j+1} + x_{i,j} + 1, & 1 \le i \le n, 1 \le j \le n-2 \\
x_{i+1,j}x_{i+2,j} + x_{i,j}x_{i+2,j} + x_{i+2,j} & \\
\qquad +x_{i,j}x_{i+1,j} + x_{i+1,j} + x_{i,j} + 1, & 1 \le j \le n, 1 \le i \le n-2 \\
1 + \sum\limits_{\substack{k=1 \\ n/2 \preceq k}}^{n} \sigma_{k,n}(x_{i,1}, \ldots, x_{i,n}), & 1 \le i \le n \\
1 + \sum\limits_{\substack{k=1 \\ n/2 \preceq k}}^{n} \sigma_{k,n}(x_{1,j}, \ldots, x_{n,j}), & 1 \le j \le n \\
\prod\limits_{k=1}^{n}(x_{i,k} + x_{j,k} + 1), & 1 \le i < j \le n \\
\prod\limits_{k=1}^{n}(x_{k,i} + x_{k,j} + 1), & 1 \le i < j \le n \\
x_{i,j}^2 + x_{i,j} & 1 \le i \le n, 1 \le j \le n.
\end{array}
$$

The polynomials of the form $x_i^2 + x_i$ are added to the systems in order to eliminate possible solutions in the set $\overline{\mathbb{F}}_2 \setminus \mathbb{F}_2$, where $\overline{\mathbb{F}}_2$ is the algebraic closure of $\mathbb{F}_2$. If the value of t entries of an $n \times n$ binary puzzle are initially given, the number of variables in the system can be reduced to $n - t$ by substituting the corresponding variables with their given values.

7.6 Polynomial Equations over $\mathbb{Q}$ (or $\mathbb{Z}$)

One of the important parameters that determines the complexity of solving system of equations is the degree of the polynomials in the system. In some cases, such as the system of equations from block ciphers (see Subsection 2.3.1) and in the proof of NP-completeness of MQ over $\mathbb{F}_2$ (Proposition 2.4), the degree of polynomials in the system can be reduced by introducing some auxiliary variables. In the case of binary puzzles, we ask if it is possible to reduce the degree of some equations without introducing additional variables in the system.

Although it is natural to express all three constraints of a binary puzzle as polynomials over $\mathbb{F}_2$, one may observe that the polynomials representing the balancedness constraint can be simplified by working over a different ring, such as $\mathbb{Q}$ or $\mathbb{Z}$.

Theorem 7.16. *Let $x_1, \ldots, x_n$ be variables that represent n cells in a single row/column of an $n \times n$ binary puzzle. The second constraint of binary puzzle is represented by the following polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$)*

$$\sum_{i=1}^{n} x_i - n/2. \tag{7.6}$$

Similarly, we can also express the distinctness constraint as polynomials over $\mathbb{Q}$ (or $\mathbb{Z}$) in the following theorem.

Theorem 7.17. *Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be variables that represent two distinct rows (or columns) of an $n \times n$ binary puzzle. The polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$) that represents the distinctness constraint of a binary puzzle is given by*

$$\prod_{i=1}^{n}(x_i + y_i - 1). \tag{7.7}$$

The remaining step is to obtain the polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$) that represents the three-value constraint. While this may not be obvious from the definition of the constraint, we can derive the polynomial representation over $\mathbb{Q}$ (or $\mathbb{Z}$) from the Boolean function (7.3) that represent the first constraint.

Theorem 7.18. *Let x, y, z be variables that represent three adjacent cells (horizontally or vertically) of an $n \times n$ binary puzzle. The three-value constraint of binary puzzle is represented by the following polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$)*

$$yz + xz - z + xy - y - x + 1 \tag{7.8}$$

Proof. Let φ_{f_1} be the pseudo Boolean function of f_1 in (7.3). The numerical normal form of f_1 over $\mathbb{Q}$ (or $\mathbb{Z}$) is the polynomial that represent the first constraint of binary puzzle. The coefficients of the NNF of f_1 can be computed using φ_{f_1} and Proposition 7.2. One can verify that the NNF of f_1 is equal to (7.8). ■

In order to eliminate superfluous solutions, i.e., solutions that do not belong to the set $\{0,1\} \subset \mathbb{Q}$ (or $\{0,1\} \subset \mathbb{Z}$), we also add polynomials of the form $x_{i,j}^2 - x_{i,j}$ to the system. Thus, the system of equations over $\mathbb{Q}$ (or $\mathbb{Z}$) that represents an $n \times n$ binary puzzle is equal to

$$\begin{aligned}
&x_{i,j+1}x_{i,j+2} + x_{i,j}x_{i,j+2} - x_{i,j+2} \\
&\qquad + x_{i,j}x_{i,j+1} - x_{i,j+1} - x_{i,j} + 1, && 1 \le i \le n, 1 \le j \le n-2 \\
&x_{i+1,j}x_{i+2,j} + x_{i,j}x_{i+2,j} - x_{i+2,j} \\
&\qquad + x_{i,j}x_{i+1,j} - x_{i+1,j} - x_{i,j} + 1, && 1 \le j \le n, 1 \le i \le n-2 \\
&\sum_{j=1}^{n} x_{i,j} - n/2, && 1 \le i \le n \\
&\sum_{i=1}^{n} x_{i,j} - n/2, && 1 \le j \le n \\
&\prod_{k=1}^{n} (x_{i,k} + x_{j,k} - 1), && 1 \le i \le n-1, i+1 \le j \le n \\
&\prod_{k=1}^{n} (x_{k,i} + x_{k,j} - 1), && 1 \le i \le n-1, i+1 \le j \le n \\
&x_{i,j}^2 - x_{i,j} && 1 \le i \le n, 1 \le j \le n.
\end{aligned}$$

The system of equations over $\mathbb{Q}$ (or $\mathbb{Z}$) of an $n \times n$ binary puzzle serves as an example where the degree of some polynomials in the system can be reduced without introducing additional variables. However, this comes at the cost of defining the polynomials over a different ring.

7.7 Solving Binary Puzzles using Gröbner Bases

In order to understand the applicability of Gröbner bases algorithm to solve binary puzzles, it is necessary to perform some practical experiments. The experiments must cover the system of equations of binary puzzle over $\mathbb{F}_2$, $\mathbb{Q}$, and $\mathbb{Z}$. We will compare the performance and memory usage of existing implementation of Gröbner bases algorithm in various computer algebra system. These results can shed some lights in understanding the cost of reducing the degree of the polynomials when defined over $\mathbb{F}_2$, at the expense of defining them over $\mathbb{Q}$ or $\mathbb{Z}$.

We ran the experiments on binary puzzles of size $n = 6, 8, 10, 12, 14$. For each n, we took the first five (5) samples of binary puzzles classified under "very hard" category from `http://www.binarypuzzle.com/`. The experiments were executed on a single core Intel(R) Xeon(R) CPU E5-4640 0 @ 2.40GHz equipped with 128GB of RAM.

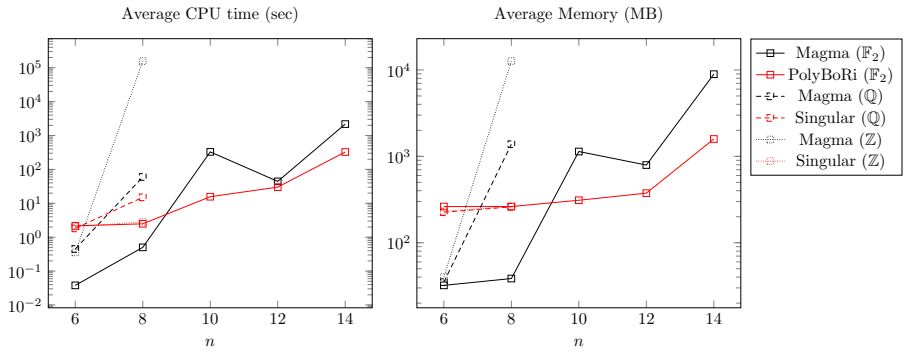


Figure 14: *Average CPU time and memory usage of various implementations of Gröbner bases algorithm to solve polynomial systems from $n \times n$ binary puzzles.*

For system of polynomials over $\mathbb{F}_2$, we compared the implementation of Gröbner bases algorithm in Magma [BCP97] and PolyBoRi/BRiAl* [BD09a]. Magma provided Gröbner bases algorithm for Boolean polynomials since version 2.15.† PolyBoRi/BRiAl is an open-source implementation of Gröbner basis algorithm with specialized data-structure for Boolean polynomials using Zero-Suppressed Binary Decision Diagram (ZDD) [Min95]. We use the implementation of PolyBoRi/BRiAl inside SageMath [The18]. The results of the experiment are available in Table 17.

For system of equations that represent 6×6 and 8×8 puzzles over $\mathbb{F}_2$, the implementation of Gröbner bases algorithm in Magma consistently solved those systems faster with less memory usage compared to PolyBoRi/BRiAl. However, PolyBoRi/BRiAl solved the systems from binary puzzles of size $n \geq 10$ much faster and with much less memory usage than Magma, often by significant factor. For instance, the system of equations from 10×10 binary puzzles were solved 1.14 up to 43.94 times faster and it used 1.52 up to 7.66 less memory than Magma (with exception on the first and the last puzzle where the memory usage of Magma is less than PolyBoRi/BRiAl). This efficiency of PolyBoRi/BRiAl is even more obvious for system of equations from 14×14 binary puzzles, where it solved the second and the third cases 418 and 226 times faster than Magma, respectively. For the same case, the efficiency of its memory usage reached a factor of 52 and 37 times less than Magma. For $n = 12$, Magma performed faster than PolyBoRi/BRiAl only for the fourth case by a factor of 1.86 but with 1.06 times more memory. The remaining four cases were solved by PolyBoRi/BRiAl by a factor ranging from 3.49 up to 10.16 times faster and up to 3.3 times less memory usage than Magma.

However, it is not possible to draw a conclusion why PolyBoRi/BRiAl performed significantly better than Magma to solve system of equations from binary

* `https://github.com/BRiAl/BRiAl`
† `http://magma.maths.usyd.edu.au/magma/handbook/text/1213#13600`

puzzles with $n \geq 10$ over $\mathbb{F}_2$. While much can be said about the internal implementation of PolyBoRi/BRiAl, this is not the case with Magma due to its closed-source nature.

For system of equations over $\mathbb{Q}$ and $\mathbb{Z}$, we compared the implementations of Gröbner bases algorithm in Magma and Singular [DGPS18]. The experiments were limited to $n = 6, 8$ and the results are given in Table 18 and Table 19. For $n \geq 10$, the computation of Magma exceeded two-weeks time for one single instance of polynomial system and some of the computation of Singular were terminated because the process ran out of memory. At this point we can already see that reducing the degree of polynomials over $\mathbb{F}_2$ at the expense of defining them as polynomials over $\mathbb{Q}$ and $\mathbb{Z}$ does not gain any practical advantage to compute the Gröbner bases of the corresponding polynomial ideal. We summarized the result in Figure 14.

Note that solving a binary puzzle using Gröbner bases algorithm is not the most efficient method to obtain a solution for a binary puzzle. In [UM17], the authors demonstrated that backtrack-based search and SAT solvers are better approach to solve a binary puzzle than using Gröbner bases algorithm. The advantage of using Gröbner bases is it allows a parametrization of the set of solutions and it helps to show which cells from a configuration of a binary puzzle that has multiple solutions.

7.8 Scope and Limitations

The scope of the modelling of binary puzzles presented in this chapter, formulated as systems of equations, does not allow a direct comparison with M4GB algorithm. This is because the current implementation of the M4GB algorithm is tailored for quadratic, dense systems of equations defined over an odd prime field or a proper extension of $\mathbb{F}_2$. In contrast, the systems of equations arising from binary puzzles are sparse, of higher degree, and defined over the binary field, the integer ring, or the field of rational number.

Furthermore, a complexity analysis of solving binary puzzles via Gröbner bases is intentionally omitted. Such analysis would only be meaningful for specific instances of the puzzle, and since different instantiations yield different systems of equations, the resulting complexity estimates would vary accordingly. Consequently, this approach would not provide a representative measure of the average computational complexity of solving binary puzles using Gröbner bases algorithm.

6×6				8×8			
PolyBoRi		Magma		PolyBoRi		Magma	
Time	Memory	Time	Memory	Time	Memory	Time	Memory
2.19 s	262.6 MB	0.04 s	32.1 MB	2.45 s	263.3 MB	1.73 s	64.1 MB
2.12 s	262.4 MB	0.06 s	32.1 MB	2.35 s	263 MB	0.18 s	32.1 MB
2.22 s	261.4 MB	0.04 s	32.1 MB	2.49 s	262.3 MB	0.14 s	32.1 MB
2.10 s	261.3 MB	0.03 s	32.1 MB	2.48 s	261.8 MB	0.32 s	32.1 MB
2.19 s	263 MB	0.02 s	32.1 MB	2.65 s	261.9 MB	0.14 s	32.1 MB

10×10			
PolyBoRi		Magma	
Time	Memory	Time	Memory
3.84 s	264.4 MB	15.45 s	250.8 MB
61.7 s	490.9 MB	1325.1 s	3761.1 MB
4.23 s	264.7 MB	51.38 s	401.5 MB
5.68 s	267.6 MB	249.56 s	1074.8 MB
2.94 s	263.9 MB	3.36 s	198.9 MB

12×12			
PolyBoRi		Magma	
Time	Memory	Time	Memory
17.34 s	320.9 MB	60.46 s	1011.6 MB
4.10 s	265.5 MB	18.01 s	545.7 MB
4.72 s	265.5 MB	36.77 s	725.5 MB
119.86 s	756.0 MB	64.13 s	804.7 MB
4.19 s	266.3 MB	42.56 s	878.3 MB

14×14			
PolyBoRi		Magma	
Time	Memory	Time	Memory
386.32 s	1659.5 MB	1238 s	6880.75 MB
11.85 s	279 MB	4962 s	14554.69 MB
9.32 s	275.3 MB	2111 s	10292.16 MB
401.83 s	2161.5 MB	903 s	4422.66 MB
823.32 s	3563.1 MB	1667 s	8539.16 MB

Table 17: *Comparison of CPU time and memory usage to solve binary puzzles using Gröbner bases algorithms for Boolean polynomials implemented in PolyBoRi and Magma.*

6×6			
Singular		Magma	
Time	Memory	Time	Memory
1.84 s	226.46 MB	0.46 s	35.4 MB
1.83 s	225.14 MB	0.46 s	38.3 MB
1.74 s	225.80 MB	0.27 s	28.9 MB
1.84 s	225.54 MB	0.63 s	38.9 MB
1.87 s	224.48 MB	0.45 s	33.6 MB

8×8			
Singular		Magma	
Time	Memory	Time	Memory
61.59 s	350.24 MB	61.84 s	2075.59 MB
5.32 s	245.12 MB	214.57 s	4033.69 MB
2.26 s	233.37 MB	4.57 s	152.12 MB
3.19 s	237.69 MB	12.19 s	488.4 MB
2.83 s	234.84 MB	5.72 s	158.35 MB

Table 18: *Comparison of CPU time and memory usage to solve binary puzzles as polynomials equations over $\mathbb{Q}$ using implementation of Gröbner bases in Singular and Magma.*

6×6			
Singular		Magma	
Time	Memory	Time	Memory
2.32 s	234.14 MB	0.38 s	43.84 MB
2.04 s	231.65 MB	0.54 s	46.43 MB
1.94 s	230.73 MB	0.24 s	30.07 MB
2.05 s	230.05 MB	0.41 s	44.64 MB
1.95 s	229.97 MB	0.22 s	35.88 MB

8×8			
Singular		Magma	
Time	Memory	Time	Memory
3.12 s	293.86 MB	778713 s	53762.45 MB
2.34 s	253.30 MB	4248.12 s	7486.61 MB
2.34 s	247.43 MB	88.04 s	203.57 MB
2.35 s	249.87 MB	336.85 s	1386.29 MB
3.98 s	266.30 MB	112.95 s	460.86 MB

Table 19: *Comparison of CPU time and memory usage to solve binary puzzles as polynomials equations over $\mathbb{Z}$ using implementation of Gröbner bases in Singular and Magma.*

8 Ideals of Vectorial Boolean Functions

Contents

The main object studied in this chapter is the multi-output Boolean functions, that is functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ for some positive integers n and m. Such functions are called *vectorial Boolean functions*. The vectorial Boolean functions arise in many aspect of symmetric-key primitives, from S-Boxes, round functions, block ciphers, hash functions, etc. Many criteria have been developed to evaluate their suitability for cryptographic purposes. For instance, to design a block cipher immune against linear cryptanalysis [Mat93] the S-Boxes used must have high nonlinearity [Nyb92]. Another criterion is related to immunity of a primitives against differential cryptanalysis [BS90], where the S-Boxes should have low differential uniformity [Nyb93]. In addition to the two previous criteria, the absence of linear structures [MS89b] is another criterion which was recently shown to be related with differential-linear cryptanalysis [CKL+19, BDKW19].

In this chapter we shall present the relation among cryptographic criteria of vectorial Boolean functions and their corresponding multivariate ideals. We focus on the (non)linearity, differential, and autocorrelation of vectorial Boolean functions. We also establish the relation between the Walsh transform as well as Fourier transform with properties of multivariate ideals. Moreover, we exhibit new algorithms to compute the bias of a linear approximation, the probability of a differential, and the autocorrelation of the component functions using a Gröbner bases algorithm.

Generous examples are provided to illustrate the important results. We provide an open-source SageMath [The18] module Gbsbox to facilitate the verification of examples and to stimulate the research on this area. The module is available for download in the following link

`https://github.com/rusydi/gbsbox`.

We begin this chapter by describing necessary terminologies, notations, and well-known results from computational commutative algebra and Boolean functions in Section 8.1. This will be followed by Section 8.2, which describes the relation between properties of a vectorial Boolean function relevant for linear cryptanalysis and its corresponding ideal. An algorithm to compute the bias of a linear approximation based on a Gröbner basis algorithm is described in the same section. Section 8.3 presents the relation between properties of a vectorial Boolean function relevant for differential cryptanalysis and its corresponding ideal. The same section also explains an algorithm to compute the probability of a differential using a Gröbner basis algorithm. The relation of the ideal of a vectorial Boolean function with the notions of autocorrelation is described in Section 8.4.

Related Work

To the best of our knowledge, the connection between computational commutative algebra and cryptographic properties of vectorial Boolean functions remains underexplored to this date. The closest related works that we found are due to Bellini, Simonetti, and Sala [BSS14] as well as the work of Bellini, Mora, and Sala [BMS16]. Both papers proposed new algorithms to compute the nonlinearity of Boolean functions using Gröbner bases algorithms, where the latter is an improvement of the former.

Our work differs from [BSS14, BMS16] in two respects. Firstly, their results as a mapping $\mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are restricted for the case $m = 1$, while the results in this chapter holds for any positive integers n, m. Secondly, the applications of Gröbner bases in [BSS14, BMS16] are limited within the context of (non)linearity, while this work proposes new applications of Gröbner bases related to the notions of (non)linearity, differential and autocorrelation.

Another related work is by Samardjiska and Gligoroski [SG14]. In their work, the authors employ (non)linearity measures that are typically used to assess the security of symmetric-key primitives to redefine certain properties of MPKCs. Their framework enables a generalization of several known attacks on MPKC schemes and offers insights into the factors that determine their effectiveness.

The differences between the results in this chapter and those in [SG14] are twofold. First, while the work in [SG14] draws upon the notions from symmetric-key cryptography to analyze MPKC schemes, our work takes the opposite direction: we apply notions from computational commutative algebra, typically used in the analysis of MPKCs, to investigate the properties of functions used in symmetric-key cryptography. Second, whereas [SG14] mainly focuses on the notion of (non)linearity, this chapter extends the discussion to include not only (non)linearity, but also differential property and autocorrelation.

8.1 Notations and Preliminaries

Let $F \subseteq \mathcal{R} = \mathbb{F}[x_1, \ldots, x_n]$ be a nonempty subset of $\mathcal{R}$. The main results in this chapter are relevant for $\mathbb{F} = \mathbb{F}_2$ but the propositions in this section are true for any field $\mathbb{F}$ or any finite field $\mathbb{F}_q$. We allow a mixed notation and write $\langle F, f\rangle$ for an ideal generated by $F \cup f$ where $f \in \mathcal{R}$. For any residue class ring $\mathcal{R}/I$ modulo an ideal $I \subseteq \mathcal{R}$, we denote by $\dim_{\mathbb{F}}(\mathcal{R}/I)$ the dimension of $\mathcal{R}/I$ as an $\mathbb{F}$-vector space. The set of zeros of $F \subseteq \mathcal{R}$ in an extension field $\mathbb{E}$ of $\mathbb{F}$ is denoted by $\mathbf{V}_{\mathbb{E}}(F) = \{\boldsymbol{v} \in \mathbb{E}^n : f(\boldsymbol{v}) = 0, \forall f \in F\}$.

Proposition 8.1. *For any ideal* $\{0\} \neq I \subset \mathcal{R}$*, we define* $\mathsf{RM}(I) = \mathsf{M}(\mathcal{R})\backslash\mathsf{LM}(I)$ *the set of* reduced monomials *w.r.t.* I*. Then*

$$\begin{aligned}\mathsf{RM}(I) &= \{\mathfrak{m} \in \mathsf{M}(\mathcal{R}) : s \nmid \mathfrak{m}, \forall s \in \mathsf{LM}(I)\}\\ &= \{\mathfrak{m} \in \mathsf{M}(\mathcal{R}) : s \nmid \mathfrak{m}, \forall s \in \mathsf{LM}(G)\}\end{aligned}$$

where G *is a Gröbner basis of* I*.*

Proof. See the proof of Lemma 6.51 in [BW93, pg. 272]. ■

Proposition 8.2. *The set* $\{\mathfrak{m} + I : \mathfrak{m} \in \mathsf{RM}(I)\}$ *is a basis of the* $\mathbb{F}$*-vector space* $\mathcal{R}/I$*.*

Proof. See the proof of Proposition 6.52 in [BW93, pg. 273]. ■

Proposition 8.3. *Let $\overline{\mathbb{F}}$ be an algebraic closure of the field $\mathbb{F}$ and assume that $I \subset \mathcal{R}$ is a zero-dimensional ideal. Then*

$$|\mathbf{V}_{\overline{\mathbb{F}}}(I)| \leq \dim_{\mathbb{F}}(\mathcal{R}/I).$$

*Moreover, if $\mathbb{F}$ has characteristic zero or is a finite field and I is a radical ideal, then the equality holds.**

Proof. See the proof of Theorem 8.32 in [BW93, pg. 348] ■

Definition 8.4. *For any polynomial ring $\mathcal{R} = \mathbb{F}_q[x_1, \ldots, x_n]$ defined over a finite field $\mathbb{F}_q$, we define the set of* field polynomials *of $\mathcal{R}$ as*

$$\mathsf{FP}(\mathcal{R}) = \{x_i^q - x_i : i = 1, \ldots, n\}.$$

Proposition 8.5. *If I is an ideal of $\mathcal{R} = \mathbb{F}_q[x_1, \ldots, x_n]$ where $\mathsf{FP}(\mathcal{R}) \subseteq I$, then the following holds*

1. *I is a zero-dimensional ideal,*
2. *I is a radical ideal,*
3. *$\mathbf{V}_{\mathbb{F}_q}(I) = \mathbf{V}_{\overline{\mathbb{F}_q}}(I)$.*

Proof. The proof of 1) follows from 5) in Proposition 3.46. The proof of 2) is immediate from Proposition 3.51. Finally, 3) is true since $\mathsf{FP}(\mathcal{R}) \subseteq I$ implies that $\mathbf{V}_{\overline{\mathbb{F}_q}}(I) \subseteq \mathbf{V}_{\mathbb{F}_q}(I)$. ■

In the pseudo-code of algorithms described in this chapter, the algorithm ReducedMonomials(G) takes a Gröbner basis G of a zero-dimensional ideal $I \subseteq \mathcal{R}$ and returns the set of reduced monomials of I. We provide the description of ReducedMonomials in Algorithm 8.1. The pseudo-code is adapted from the RedTerms algorithm in [BW93, pg. 424]. The algorithm maintain the set R of intermediate reduced monomials. In each iteration i, it multiplies all monomials in R by $x_i, x_i^2, \ldots, x_i^{k_i}$. The resulting non-reducible monomial product is then added to R.

For the rest of this chapter, we will work with the binary field $\mathbb{F}_2$. For any $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{F}_2^n$, we sometimes write $\boldsymbol{v}$ as an integer $\sum_{i=1}^{n} v_i 2^{i-1}$ in hexadecimal format using teletype font family (e.g, $\boldsymbol{v} = (0, 1, 1, 1) \in \mathbb{F}_2^4$ can be written as `E`). The Hamming weight of $\boldsymbol{v}$ is denoted by $\mathsf{wt}(\boldsymbol{v})$ and the concatenation of $\boldsymbol{v}$ with $\boldsymbol{w} = (w_1, \ldots, w_m) \in \mathbb{F}_2^m$ is denoted by $(\boldsymbol{v} \parallel \boldsymbol{w}) = (v_1, \ldots, v_n, w_1, \ldots, w_m)$. For any Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$, we denote the $\mathbb{Z}$-valued function of f as $\widehat{f}$. We also define the Hamming weight of a Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ as $\mathsf{wt}(f) = |\{\boldsymbol{v} \in \mathbb{F}_2^n : f(\boldsymbol{v}) \neq 0\}|$. For any $\boldsymbol{b} \in \mathbb{F}_2^m$ and $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$, we define $S_{\boldsymbol{b}}(x) = \boldsymbol{b} \cdot S(x)$ as a component functions of S, where "$\cdot$" denotes the dot product vector. We call the function S_0 the trivial component function of S. For any $\boldsymbol{e}_j \in \mathbb{F}_2^m$, where $\boldsymbol{e}_j$ denotes a vector with a 1 in its j-th coordinate and 0 otherwise, we call $S_{\boldsymbol{e}_j}$ the j-th *coordinate function* of S.

* This proposition can be generalized to any perfect field $\mathbb{F}$ (see Definition 3.52 for the definition of perfect field).

Input: A Gröbner basis G of a zero-dimensional ideal $I \subseteq \mathbb{F}[x_1, \ldots, x_n]$
Output: The set $\mathsf{RM}(I)$ of reduced monomials of I

$R \leftarrow \{1\}$
for $i = 1$ **to** n **do**
  $M \leftarrow R$
  $k_i \leftarrow \max(\{e_i : x_1^{e_1} \cdots x_i^{e_i} \cdots x_n^{e_n} \in \mathsf{LM}(G)\})$
  while $M \neq \{\}$ **do**
    $\mathfrak{m} \leftarrow \textsc{Select}(M)$
    $M \leftarrow M \setminus \{\mathfrak{m}\}$
    for $l = 1$ **to** k_i **do**
      $\mathfrak{m} \leftarrow \mathfrak{m} \cdot x_i$
      if $\forall g \in G \ : \ \mathsf{LM}(g) \nmid \mathfrak{m}$ **then**
        $R \leftarrow R \cup \{\mathfrak{m}\}$
return R

Algorithm 8.1: Algorithm ReducedMonomials.

Definition 8.6. *Let f be an n-variable Boolean function. The Fourier transform $\mathcal{F}_f : \mathbb{F}_2^n \mapsto \mathbb{Z}$ and the Walsh transform $\mathcal{W}_f : \mathbb{F}_2^n \mapsto \mathbb{Z}$ of f are defined as*

$$\mathcal{F}_f(\boldsymbol{u}) = \sum_{\boldsymbol{v} \in \mathbb{F}_2^n} \widehat{f}(\boldsymbol{v}) \cdot (-1)^{\boldsymbol{u} \cdot \boldsymbol{v}}, \qquad \mathcal{W}_f(\boldsymbol{u}) = \sum_{\boldsymbol{v} \in \mathbb{F}_2^n} (-1)^{\widehat{f}(\boldsymbol{v})} \cdot (-1)^{\boldsymbol{u} \cdot \boldsymbol{v}}.$$

Proposition 8.7. *Let f be an n-variable Boolean function. We have the following relation*

$$\mathcal{F}_f(\boldsymbol{u}) = 2^{n-1}\delta(\boldsymbol{u}) - \frac{1}{2} \cdot \mathcal{W}_f(\boldsymbol{u})$$

where $\delta(\boldsymbol{u}) = 1$ if $\boldsymbol{u} = \boldsymbol{0}$ and 0 otherwise.

Proof. See the proof of Lemma 2.9 in [CS09, pg. 9]. ■

The primary theme of this chapter is the relation between cryptographic properties of S and ideals in multivariate polynomial ring. The variables of the input and output of S are denoted using x_i, y_j respectively. A system of equations that defines S is given by the set

$$F_S = \{y_j + h_j(x_1, \ldots, x_n) : 1 \leq j \leq m\} \subset \mathbb{F}_2[x_1, \ldots, x_n, y_1, \ldots, y_m]$$

where $h_j(x_1, \ldots, x_n)$ is the algebraic normal form of $S_{\boldsymbol{e}_j}$. We refer to the set F_S as the *algebraic normal form of S*.

Throughout this chapter, we also provide examples to illustrate the main results. All examples use the mapping defined in Table 20 and its algebraic normal form consists of the following polynomials

$$y_1 + x_1x_2x_3 + x_1x_3 + x_1 + x_2 + x_3x_4,$$
$$y_2 + x_1x_2x_4 + x_1x_2 + x_1x_3x_4 + x_1x_3 + x_1 + x_2x_3x_4 + x_2x_4 + x_3 + x_4,$$
$$y_3 + x_1x_2x_4 + x_1x_2 + x_1x_3 + x_2x_3x_4 + x_2x_3 + x_2 + x_3 + x_4,$$
$$y_4 + x_1x_2x_3 + x_1x_2 + x_1x_3x_4 + x_1x_3 + x_2x_3 + x_3x_4 + x_4.$$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	0	3	5	8	6	A	F	4	E	D	9	2	1	7	C	B

Table 20: *An example of* $S : \mathbb{F}_2^4 \mapsto \mathbb{F}_2^4$.

8.2 Multivariate Ideals and Linear Property

In this section we shall prove the result stated in Theorem 1.3. The result establishes a relation between ideals in multivariate polynomial ring and properties of vectorial Boolean functions relevant for *linear cryptanalysis* [Mat93]. Linear cryptanalysis studies linear relations among parity bits of the plaintext, the ciphertext, and the secret key that holds with a certain probability. If there exists such relation with high probability, an attacker can estimate the parity bit of the key using the parity bits of known plaintexts and their corresponding ciphertexts. Formally, let $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \mapsto \mathbb{F}_2^n$ where $\mathbb{F}_2^k$ is the key space and for any fix $\boldsymbol{k} \in \mathbb{F}_2^k$ we define $E_{\boldsymbol{k}}(x) = E(x, \boldsymbol{k})$. The goal of linear cryptanalysis is to find a linear expression of the following form

$$\boldsymbol{a} \cdot \boldsymbol{v} + \boldsymbol{b} \cdot E_{\boldsymbol{k}}(\boldsymbol{v}) = \boldsymbol{c} \cdot \boldsymbol{k} \qquad \boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^n, \boldsymbol{c} \in \mathbb{F}_2^k$$

that holds with probability $p \neq 1/2$ for randomly given plaintext $\boldsymbol{v}$ and its corresponding ciphertext $E_{\boldsymbol{k}}(\boldsymbol{v})$. Clearly, the effectiveness of linear cryptanalysis depends on the magnitude of $|p - 1/2|$. Once such approximation is found, it is possible to determine the value of $\boldsymbol{c} \cdot \boldsymbol{k}$ based on the maximum likelihood method (see Algorithm 1 of [Mat93]).

The probability of a linear approximation on $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ with input mask $\boldsymbol{a} \in \mathbb{F}_2^n$ and output mask $\boldsymbol{b} \in \mathbb{F}_2^m$ is equal to

$$\mathsf{Pr}[\boldsymbol{a} \cdot x = \boldsymbol{b} \cdot S(x)] = 2^{-n} \cdot |\{\boldsymbol{v} \in \mathbb{F}_2^n : \boldsymbol{a} \cdot \boldsymbol{v} = \boldsymbol{b} \cdot S(\boldsymbol{v})\}|.$$

and we refer to the value $\mathsf{Pr}[\boldsymbol{a} \cdot x = \boldsymbol{b} \cdot S(x)] - 1/2$ as the *bias* of the linear approximation. If n, m are relatively small, we can enumerate the bias of all possible linear approximations on S and store the corresponding numerator in a $2^n \times 2^m$ *linear approximation table* of S. The entry at row $\boldsymbol{a} \in \mathbb{F}_2^n$ and column $\boldsymbol{b} \in \mathbb{F}_2^m$ of the linear approximation table of S is defined as

$$\lambda_S(\boldsymbol{a}, \boldsymbol{b}) = |\{(\boldsymbol{v} \parallel S(\boldsymbol{v})) \in \mathbb{F}_2^{n+m} : \boldsymbol{a} \cdot \boldsymbol{v} = \boldsymbol{b} \cdot S(\boldsymbol{v})\}| - 2^{n-1}.$$

An example of a linear approximation table is provided in Table 21.

Definition 8.8 (Linear ideal)**.** *Let* $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ *and let* $\mathcal{R}$ *be the polynomial ring* $\mathbb{F}_2[x_1, \ldots, x_n, y_1, \ldots, y_m]$. *For any* $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{F}_2^n, \boldsymbol{b} = (b_1, \ldots, b_m) \in \mathbb{F}_2^m$ *we define the linear ideal* $\lambda I_S(\boldsymbol{a}, \boldsymbol{b}) \subseteq \mathcal{R}$ *of* S *with input mask* $\boldsymbol{a}$ *and output mask* $\boldsymbol{b}$ *as*

$$\lambda I_S(\boldsymbol{a}, \boldsymbol{b}) = \left\langle F_S, \sum_{i=1}^{n} a_i x_i + \sum_{j=1}^{m} b_j y_j, \mathsf{FP}(\mathcal{R}) \right\rangle$$

where $F_S \subset \mathcal{R}$ *is the algebraic normal form of* S.

Theorem 8.9. *Let* $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ *and* $\mathcal{R} = \mathbb{F}_2[x_1, \ldots, x_n, y_1, \ldots, y_m]$. *The bias of a linear approximation on* S *with input mask* $\boldsymbol{a} \in \mathbb{F}_2^n$ *and output mask* $\boldsymbol{b} \in \mathbb{F}_2^m$ *is equal to*

$$2^{-n} \cdot (\dim_{\mathbb{F}_2}(\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) - 2^{n-1}).$$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	2	-2	-2	-2	4	-	-	-	2	-2	2	2	-	4
2	-	-	-2	-2	-	-	2	2	2	-2	-	4	2	-2	4	-
3	-	4	-	-	2	-2	2	2	2	2	2	-2	-	-	-	-4
4	-	-	2	-2	2	2	-	4	-	-	-2	2	2	2	-4	-
5	-	-	-	-	-	-	-	-	-	-	4	4	-4	4	-	-
6	-	-	-	4	-2	-2	-2	2	-2	2	2	2	4	-	-	-
7	-	4	-2	2	-	4	2	-2	-2	-2	-	-	2	2	-	-
8	-	2	-	2	-	2	-	2	2	-	2	-	-2	-4	-2	4
9	-	-2	-2	-	-2	4	-	2	2	4	-	-2	-	2	2	-
A	-	2	-2	-	4	-2	-2	-	-	2	-2	-	-	2	2	4
B	-	2	4	2	-2	-	2	-	-	2	-4	2	-2	-	2	-
C	-	-2	2	4	2	-	-	2	2	-4	-	-2	-	2	2	-
D	-	2	4	-2	-	2	-4	-2	2	-	2	-	2	-	2	-
E	-	-2	-	2	2	-	2	-4	4	2	-	2	2	-	-2	-
F	-	-2	2	-	4	2	2	-	-4	2	2	-	-	-2	2	-

Table 21: *Linear approximation table of S defined in Table 20. The entries denoted by '-' are equal to zero.*

Proof. Let $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$ and $\boldsymbol{b} = (b_1, \ldots, b_m) \in \mathbb{F}_2^m$. Suppose that the system of equations $F = F_S \cup \{\sum_{i=1}^n a_i x_i + \sum_{j=1}^m b_j y_j\} \cup \mathsf{FP}(\mathcal{R})$ has a solution in $\mathbb{F}_2^{n+m}$. We claim that

$$\mathbf{V}_{\mathbb{F}_2}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) = \{(\boldsymbol{v} \parallel \boldsymbol{w}) \in \mathbb{F}_2^{n+m} : \boldsymbol{v} \in \mathbb{F}_2^n, \boldsymbol{w} = S(\boldsymbol{v}) \text{ and } \sum_{i=1}^n a_i v_i + \sum_{j=1}^m b_j w_j = 0\}.$$

Let $\boldsymbol{u} = (u_1, u_2, \ldots, u_{n+m}) \in \mathbf{V}_{\mathbb{F}_2}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b}))$. Since $\mathbf{V}_{\mathbb{F}_2}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) = \mathbf{V}_{\mathbb{F}_2}(F)$, then $f(\boldsymbol{u}) = 0$ for all $f \in F$. This implies that $(u_{n+1}, \ldots, u_{n+m}) = S(u_1, \ldots, u_n)$ (by the definition of F_S) and $\sum_{i=1}^n a_i u_i + \sum_{j=1}^m b_j u_{n+j} = 0$. The converse inclusion is obvious since the condition $\boldsymbol{w} = S(\boldsymbol{v})$ implies $f(\boldsymbol{v} \parallel \boldsymbol{w}) = 0$ for all $f \in F_S$ and clearly $(\boldsymbol{v} \parallel \boldsymbol{w})$ is a solution for the polynomial $\sum_{i=1}^n a_i x_i + \sum_{j=1}^m b_j y_j$. Thus, $(\boldsymbol{v} \parallel \boldsymbol{w}) \in \mathbf{V}_{\mathbb{F}_2}(F) = \mathbf{V}_{\mathbb{F}_2}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b}))$.

Since $\mathsf{FP}(\mathcal{R}) \subset \lambda I_S(\boldsymbol{a}, \boldsymbol{b})$, by 1) and 2) of Proposition 8.5, the ideal $\lambda I_S(\boldsymbol{a}, \boldsymbol{b})$ is zero-dimensional and radical. We then have the following relation

$$|\mathbf{V}_{\overline{\mathbb{F}_2}}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b}))| = \dim_{\mathbb{F}_2}(\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) = |\mathbf{V}_{\mathbb{F}_2}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b}))|.$$

where the first equality is implied by Proposition 8.3 and the second equality is implied by 3) of Proposition 8.5. Therefore we have

$$\lambda_S(\boldsymbol{a}, \boldsymbol{b}) = \dim_{\mathbb{F}_2}(\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) - 2^{n-1} \tag{8.1}$$

and the bias of the linear approximation is equal to $2^{-n} \cdot (\dim_{\mathbb{F}_2}(\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) - 2^{n-1})$. ∎

In the following lemma, for a given $\boldsymbol{a} \in \mathbb{F}_2^n$, we denote by $\ell_{\boldsymbol{a}}$ the n-variable Boolean function $\ell_{\boldsymbol{a}}(x) = \boldsymbol{a} \cdot x$ and for any n-variable Boolean functions f, g we define $d(f, g) = |\{\boldsymbol{x} \in \mathbb{F}_2^n : f(\boldsymbol{x}) \neq g(\boldsymbol{x})\}|$ to be the *Hamming distance* of f and g.

Lemma 8.10. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. We have the following equality*

$$\mathcal{W}_{S_{\boldsymbol{b}}}(\boldsymbol{a}) = 2^n - 2 \cdot d(S_{\boldsymbol{b}}, \ell_{\boldsymbol{a}}).$$

Proof. Recall that $\mathcal{W}_{S_{\boldsymbol{b}}}(\boldsymbol{a}) = \sum_{\boldsymbol{x}\in\mathbb{F}_2^n}(-1)^{\widehat{S_{\boldsymbol{b}}}(\boldsymbol{x})}(-1)^{\widehat{\ell_{\boldsymbol{a}}}(\boldsymbol{x})}$. For any $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $S_{\boldsymbol{b}}(\boldsymbol{x}) = \ell_{\boldsymbol{a}}(\boldsymbol{x})$ we have $(-1)^{\widehat{S_{\boldsymbol{b}}}(\boldsymbol{x})}(-1)^{\widehat{\ell_{\boldsymbol{a}}}(\boldsymbol{x})} = 1$ and -1 otherwise. The number of -1 in the summation is equal to $|\{\boldsymbol{x} \in \mathbb{F}_2^n : S_{\boldsymbol{b}}(\boldsymbol{x}) \neq \ell_{\boldsymbol{a}}(\boldsymbol{x})\}| = d(S_{\boldsymbol{b}}, \ell_{\boldsymbol{a}})$. On the other hand, the number of $+1$ in the summation is equal to $2^n - d(S_{\boldsymbol{b}}, \ell_{\boldsymbol{a}})$. Therefore $\mathcal{W}_{S_{\boldsymbol{b}}}(\boldsymbol{a}) = 2^n - 2 \cdot d(S_{\boldsymbol{b}}, \ell_{\boldsymbol{a}})$. ■

Theorem 8.11. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and $\mathcal{R} = \mathbb{F}_2[x_1, \ldots, x_n, y_1, \ldots, y_m]$. For any $\boldsymbol{b} \in \mathbb{F}_2^m$, we have the following*

$$\mathcal{W}_{S_{\boldsymbol{b}}}(\boldsymbol{a}) = 2 \cdot \dim_{\mathbb{F}_2}(\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) - 2^n, \tag{8.2}$$

$$\mathcal{F}_{S_{\boldsymbol{b}}}(\boldsymbol{a}) = 2^{n-1}(\delta(\boldsymbol{a}) + 1) - \dim_{\mathbb{F}_2}(\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) \tag{8.3}$$

where $\delta(\boldsymbol{a}) = 1$ if $\boldsymbol{a} = \boldsymbol{0}$ and 0 otherwise.

Proof. By the definition of $\lambda_S(\boldsymbol{a}, \boldsymbol{b})$, we can express it as

$$\begin{aligned}\lambda_S(\boldsymbol{a}, \boldsymbol{b}) &= |\{(\boldsymbol{v} \parallel S(\boldsymbol{v})) \in \mathbb{F}_2^{n+m} : \boldsymbol{a} \cdot \boldsymbol{v} = \boldsymbol{b} \cdot S(\boldsymbol{v})\}| - 2^{n-1} \\ &= 2^n - d(S_{\boldsymbol{b}}, \ell_{\boldsymbol{a}}) - 2^{n-1} \\ &= 2^{n-1} - d(S_{\boldsymbol{b}}, \ell_{\boldsymbol{a}})\end{aligned}$$

and by Lemma 8.10 we have $\mathcal{W}_{S_{\boldsymbol{b}}}(\boldsymbol{a}) = 2 \cdot \lambda_S(\boldsymbol{a}, \boldsymbol{b})$. From this, the proof of (8.2) is immediate from (8.1), i.e.,

$$\mathcal{W}_{S_{\boldsymbol{b}}}(\boldsymbol{a}) = 2 \cdot \lambda_S(\boldsymbol{a}, \boldsymbol{b}) = 2 \cdot \dim_{\mathbb{F}_2}(\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) - 2^n.$$

The proof of (8.3) is a straightforward implication of Proposition 8.7 and (8.2). ■

Input: A vectorial Boolean function $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$
Input: An input mask $\boldsymbol{a} \in \mathbb{F}_2^n$
Input: An output mask $\boldsymbol{b} \in \mathbb{F}_2^m$
Output: The bias of the linear approximation $\boldsymbol{a} \cdot x = \boldsymbol{b} \cdot S(x)$.
1 $\mathcal{R} \leftarrow \mathbb{F}_2[x_1, \ldots, x_n, y_1, \ldots, y_m]$
2 $F_S \leftarrow$ Algebraic Normal Form of S
3 $\lambda I_S(\boldsymbol{a}, \boldsymbol{b}) \leftarrow \langle F_S, \sum_{i=1}^n a_i x_i + \sum_{j=1}^m b_j y_j, \mathsf{FP}(\mathcal{R})\rangle$
4 $G \leftarrow$ GröbnerBasis($\lambda I_S(\boldsymbol{a}, \boldsymbol{b})$)
5 $\mathsf{RM}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b})) \leftarrow$ ReducedMonomials(G)
6 **return** $2^{-n} \cdot (|\mathsf{RM}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b}))| - 2^{n-1})$

Algorithm 8.2: An algorithm to compute the bias of a linear approximation on S using a Gröbner bases algorithm.

Theorem 8.12. *Given $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$, $\boldsymbol{a} \in \mathbb{F}_2^n$ and $\boldsymbol{b} \in \mathbb{F}_2^m$, Algorithm 8.2 computes the bias of a linear approximation on S with input mask $\boldsymbol{a}$ and output mask $\boldsymbol{b}$.*

Proof. Termination: Since each step in line 1-4 terminates, then eventually the algorithm terminates.*

* The termination of a Gröbner bases algorithm is due to the Noetherianity of $\mathcal{R}$ (see for example the proof of Theorem 2 in [CLO15, pg. 91]) and the termination of ReducedMonomials is due to the fact that the ideal $\lambda I_S(\boldsymbol{a}, \boldsymbol{b})$ is zero-dimensional (see Proposition 8.1 and 4) of Proposition 3.46).

Correctness: By Proposition 8.2, the set $\{\mathfrak{m} + \lambda I_S(\boldsymbol{a}, \boldsymbol{b}) : \mathfrak{m} \in \mathsf{RM}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b}))\}$ is a basis of the $\mathbb{F}_2$-vector space $\mathcal{R}/\lambda I_S(\boldsymbol{a}, \boldsymbol{b})$. It follows from Theorem 8.9, that the bias of the linear approximation $\boldsymbol{a} \cdot x = \boldsymbol{b} \cdot S(x)$ is equal to $2^{-n} \cdot (|\mathsf{RM}(\lambda I_S(\boldsymbol{a}, \boldsymbol{b}))| - 2^{n-1})$. ■

Example 8.13. *Let S be the mapping defined in Table 20. Let $\boldsymbol{a} = \boldsymbol{F} = (1,1,1,1), \boldsymbol{b} = \boldsymbol{\mathcal{S}} = (0,0,0,1)$ and $I = \lambda I_S(\boldsymbol{a}, \boldsymbol{b})$. The reduced Gröbner basis of the ideal I w.r.t. degree-reverse lexicographic ordering consists of the following polynomials*

$$\begin{array}{lll} y_1^2 + y_1, & y_2y_4 + y_2, & x_3 + y_1 + y_2 + y_4, \\ y_1y_2, & y_4^2 + y_4, & x_4 + y_1 + y_4, \\ y_2^2 + y_2, & x_1, & y_3 + y_4, \\ y_1y_4, & x_2 + y_2 + y_4. & \end{array}$$

The basis of the $\mathbb{F}_2$-vector space $\mathcal{R}/I$ is the following

$$\{y_4 + I, y_2 + I, y_1 + I, 1 + I\}.$$

Therefore $\lambda_S(\boldsymbol{F}, \boldsymbol{\mathcal{S}}) = \dim_{\mathbb{F}_2}(\mathcal{R}/I) - 2^3 = -4$ (see also Table 21).

8.3 Multivariate Ideals and Differential Property

In this section we present relations between ideals in multivariate polynomial ring and properties of vectorial Boolean functions that are relevant for *differential cryptanalysis*. Differential cryptanalysis is one class of attacks applicable to wide-range of symmetric-key primitives, particularly block ciphers and hash functions. The attack studies how a difference in two inputs to a vectorial Boolean function affects differences of their corresponding outputs. The common difference operation is bitwise exclusive-or (XOR) operation, which corresponds to vector addition in $\mathbb{F}_2^n$. This is also the difference operation considered throughout this chapter.

Let $\boldsymbol{\alpha} \in \mathbb{F}_2^n, \boldsymbol{\beta} \in \mathbb{F}_2^m$ be differences in the input and output of $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ respectively. We refer to the pair $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as a *differential* of S and its probability is defined as

$$\mathsf{Pr}[\boldsymbol{\alpha} \xrightarrow{S} \boldsymbol{\beta}] = \frac{|\{\boldsymbol{v} \in \mathbb{F}_2^n : S(\boldsymbol{v} + \boldsymbol{\alpha}) + S(\boldsymbol{v}) = \boldsymbol{\beta}\}|}{2^n} = \frac{\delta_S(\boldsymbol{\alpha}, \boldsymbol{\beta})}{2^n}.$$

Note that for small n, m, the values $(\delta_S(\boldsymbol{\alpha}, \boldsymbol{\beta}))_{\boldsymbol{\alpha} \in \mathbb{F}_2^n, \boldsymbol{\beta} \in \mathbb{F}_2^m}$ are typically stored in a two-dimensional array called the *difference distribution table* of S. An example of a difference distribution table is provided in Table 22. Another relevant parameter to determine the resistance of a vectorial Boolean function against differential cryptanalysis is *differential uniformity* [Nyb93].

Definition 8.14. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. The function S has differential uniformity equal to δ_S if*

$$\delta_S = \max_{0 \neq \boldsymbol{\alpha} \in \mathbb{F}_2^n, \boldsymbol{\beta} \in \mathbb{F}_2^m} \delta_S(\boldsymbol{\alpha}, \boldsymbol{\beta}).$$

In order to study the properties relevant for differential cryptanalysis, we first need to construct the required polynomial ring. Recall that in differential cryptanalysis, we consider two inputs to a vectorial Boolean function $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. Let $x'_1, \ldots, x'_n$ and $x''_1, \ldots, x''_n$ be two sets of variables that represent two

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
1	–	–	–	4	–	–	2	2	–	–	–	4	2	2	–	–
2	–	–	–	–	–	2	–	2	–	2	–	2	2	2	2	2
3	–	–	2	–	2	2	2	–	2	–	2	2	2	–	–	–
4	–	–	–	–	–	2	2	–	–	4	4	–	2	–	–	2
5	–	2	2	–	–	2	–	2	–	2	2	–	2	–	2	–
6	–	–	4	2	–	2	2	2	2	–	–	–	–	–	–	2
7	–	2	–	2	2	2	–	–	–	–	–	–	2	–	4	2
8	–	–	–	2	–	–	–	2	–	–	2	–	2	2	4	2
9	–	4	–	–	2	–	–	2	2	–	–	2	–	4	–	–
A	–	4	–	2	–	2	–	–	–	2	2	2	–	–	2	–
B	–	–	2	–	–	2	4	–	4	–	2	–	–	2	–	–
C	–	2	–	2	2	–	4	2	2	2	–	–	–	–	–	–
D	–	–	2	–	4	–	–	2	–	–	–	2	–	4	2	–
E	–	2	–	–	2	–	–	–	4	2	–	–	2	–	–	4
F	–	–	4	2	2	–	–	–	–	2	2	2	–	–	–	2

Table 22: *Difference distribution table of S defined in Table 20. The entries denoted by '–' are equal to zero.*

inputs to S. Their respective output are denoted by variables $y'_1, \ldots, y'_m$ and $y''_1, \ldots, y''_m$. Thus, for notions related to differential cryptanalysis, we will use the following ring

$$\mathcal{R} = \mathbb{F}_2[x'_1, \ldots, x'_n, y'_1, \ldots, y'_m, x''_1, \ldots, x''_n, y''_1, \ldots, y''_m]. \tag{8.4}$$

However, we stress that the results of this chapter are independent from the choice of variable ordering and the monomial ordering in $\mathcal{R}$.

Definition 8.15 (Differential ideal)**.** *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and let $\mathcal{R}$ be a polynomial ring defined in* (8.4)*. For any $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_2^n, \boldsymbol{\beta} = (\beta_1, \ldots, \beta_m) \in \mathbb{F}_2^m$ we define the differential ideal $\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}) \subseteq \mathcal{R}$ of S w.r.t. the differential $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as*

$$\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \langle F'_S, F''_S, \{x'_i + x''_i + \alpha_i \ : 1 \le i \le n\}, \\ \{y'_j + y''_j + \beta_j : 1 \le j \le m\}, \mathsf{FP}(\mathcal{R})\rangle$$

where F'_S and F''_S are the sets of polynomials that represent S in variables x'_i, y'_j and x''_i, y''_j respectively.

Theorem 8.16. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and let $\mathcal{R}$ be a polynomial ring defined in* (8.4)*. The probability of a differential $(\boldsymbol{\alpha}, \boldsymbol{\beta}) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ is equal to*

$$\mathsf{Pr}[\boldsymbol{\alpha} \xrightarrow{S} \boldsymbol{\beta}] = \frac{\dim_{\mathbb{F}_2}(\mathcal{R}/\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}))}{2^n}.$$

Proof. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_2^n$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m) \in \mathbb{F}_2^m$. Suppose that the following system of equations

$$F = F'_S \cup F''_S \cup \{x'_i + x''_i + \alpha_i : 1 \le i \le n\} \cup \{y'_j + y''_j + \beta_j : 1 \le j \le m\} \cup \mathsf{FP}(\mathcal{R})$$

has a solution in $\mathbb{F}_2^{2(n+m)}$. Let $\Delta_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$ be the following set

$$\Delta_S(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \{(\boldsymbol{v} \parallel S(\boldsymbol{v}) \parallel \boldsymbol{v} + \boldsymbol{\alpha} \parallel S(\boldsymbol{v} + \boldsymbol{\alpha})) : \boldsymbol{v} \in \mathbb{F}_2^n, S(\boldsymbol{v} + \boldsymbol{\alpha}) + S(\boldsymbol{v}) = \boldsymbol{\beta}\}.$$

We will first show that $\mathbf{V}_{\mathbb{F}_2}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})) = \Delta_S(\boldsymbol{\alpha},\boldsymbol{\beta})$.

Let $\boldsymbol{u} = (u_1,\ldots,u_{2(n+m)}) \in \mathbf{V}_{\mathbb{F}_2}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta}))$. Since $\mathbf{V}_{\mathbb{F}_2}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})) = \mathbf{V}_{\mathbb{F}_2}(F)$ then $f(\boldsymbol{u}) = 0$ for all $f \in F$. Thus, $(u_{n+1},\ldots,u_{n+m}) = S(u_1,\ldots,u_n)$ by the definition of F'_S, $(u_{n+m+1},\ldots,u_{2n+m}) = (u_1+\alpha_1,\ldots,u_n+\alpha_n)$ due to $u_i + u_{n+m+i} + \alpha_i = 0$ for $1 \le i \le n$, and $(u_{2n+m+1},\ldots,u_{2(n+m)}) = S(u_{n+m+1},\ldots,u_{2n+m})$ by the definition of F''_S such that $u_{n+j}+u_{2n+m+j}+\beta_j = 0$ for $1 \le j \le m$. Conversely, let $(\boldsymbol{v} \parallel S(\boldsymbol{v}) \parallel \boldsymbol{v}+\boldsymbol{\alpha} \parallel S(\boldsymbol{v}+\boldsymbol{\alpha})) \in \Delta_S(\boldsymbol{\alpha},\boldsymbol{\beta})$ where $\boldsymbol{v} \in \mathbb{F}_2^n$. Clearly $f'(\boldsymbol{v} \parallel S(\boldsymbol{v})) = 0$ for all $f' \in F'_S$ and $f''(\boldsymbol{v}+\boldsymbol{\alpha} \parallel S(\boldsymbol{v}+\boldsymbol{\alpha})) = 0$ for all $f'' \in F''_S$. Similarly, $(\boldsymbol{v} \parallel \boldsymbol{v}+\boldsymbol{\alpha})$ is a solution of $x'_i + x''_i + \alpha_i$ for $1 \le i \le n$ and $(S(\boldsymbol{v}) \parallel S(\boldsymbol{v}+\boldsymbol{\alpha}))$ is also a solution for $y'_j + y''_j + \beta_j = 0$ for $1 \le j \le m$ by the condition $S(\boldsymbol{v}+\boldsymbol{\alpha}) + S(\boldsymbol{v}) = \boldsymbol{\beta}$. Thus $(\boldsymbol{v} \parallel S(\boldsymbol{v}) \parallel \boldsymbol{v}+\boldsymbol{\alpha} \parallel S(\boldsymbol{v}+\boldsymbol{\alpha})) \in \mathbf{V}_{\mathbb{F}_2}(F) = \mathbf{V}_{\mathbb{F}_2}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta}))$.

Since $\mathsf{FP}(\mathcal{R}) \subset \delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})$, by 1) and 2) of Proposition 8.5, the ideal $\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})$ is zero-dimensional and radical. We then have the following relation

$$|\mathbf{V}_{\overline{\mathbb{F}_2}}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta}))| = \dim_{\mathbb{F}_2}(\mathcal{R}/\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})) = |\mathbf{V}_{\mathbb{F}_2}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta}))|$$

where the first equality is an implication of Proposition 8.3 and the second equality is an implication of 3) in Proposition 8.5. Since it is trivial to see that $\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta}) = |\Delta_S(\boldsymbol{\alpha},\boldsymbol{\beta})|$, therefore we have the following

$$\delta_S(\boldsymbol{\alpha},\boldsymbol{\beta}) = \dim_{\mathbb{F}_2}(\mathcal{R}/\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})). \tag{8.5}$$

■

Remark 8.17. *From* (8.5) *and Definition 8.14, computing the differential uniformity of S is equivalent to finding an $\mathbb{F}_2$-vector space $\mathcal{R}/\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})$, for $\boldsymbol{0} \neq \boldsymbol{\alpha} \in \mathbb{F}_2^n$ and $\boldsymbol{\beta} \in \mathbb{F}_2^m$, of maximum dimension.*

In the following, we shall apply the result from Theorem 8.16 to compute the probability of a differential using a Gröbner basis algorithm.

Theorem 8.18. *Given $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and a differential $(\boldsymbol{\alpha},\boldsymbol{\beta}) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$, Algorithm 8.3 computes the differential probability of $(\boldsymbol{\alpha},\boldsymbol{\beta})$.*

Input: A vectorial Boolean function $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$
Input: A differential $(\boldsymbol{\alpha},\boldsymbol{\beta}) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$
Output: The probability $\mathsf{Pr}[\boldsymbol{\alpha} \to^S \boldsymbol{\beta}]$
1 $\mathcal{R} \leftarrow \mathbb{F}_2[x'_1,\ldots,x'_n,y'_1,\ldots,y'_m,x''_1,\ldots,x''_n,y''_1,\ldots,y''_m]$
2 $F'_S \leftarrow$ Algebraic Normal Form of S in $x'_1,\ldots,x'_n,y'_1,\ldots,y'_m$
3 $F''_S \leftarrow$ Algebraic Normal Form of S in $x''_1,\ldots,x''_n,y''_1,\ldots,y''_m$
4 $A \leftarrow \{x'_i + x''_i + \alpha_i : 1 \le i \le n\}$
5 $B \leftarrow \{y'_j + y''_j + \beta_j : 1 \le j \le m\}$
6 $\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta}) \leftarrow \langle F'_S, F''_S, A, B, \mathsf{FP}(\mathcal{R})\rangle$
7 $G \leftarrow$ GröbnerBasis($\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})$)
8 $\mathsf{RM}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta})) \leftarrow$ ReducedMonomials(G)
9 **return** $|\mathsf{RM}(\delta I_S(\boldsymbol{\alpha},\boldsymbol{\beta}))|/2^n$

Algorithm 8.3: An algorithm to compute the probability of a differential on S using a Gröbner bases algorithm.

Proof. Termination : The termination of the above algorithm is obvious since the computation of algebraic normal form of S, Gröbner basis of $\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$, and the ReducedMonomials terminates where the latter is due to the ideal $\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$ being a zero-dimensional.
Correctness : By Proposition 8.2, the set $\{\mathfrak{m} + \delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}) : \mathfrak{m} \in \mathsf{RM}(\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}))\}$ is a basis of the $\mathbb{F}_2$-vector space $\mathcal{R}/\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$ and clearly by Theorem 8.16, $\mathsf{Pr}[\boldsymbol{\alpha} \xrightarrow{S} \boldsymbol{\beta}] = |\mathsf{RM}(\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}))|/2^n$. ■

Example 8.19. *Let S be the mapping defined in Table 20. Let $\boldsymbol{\alpha} = \boldsymbol{6} = (0,1,1,0) \in \mathbb{F}_2^4, \boldsymbol{\beta} = \boldsymbol{2} = (0,1,0,0) \in \mathbb{F}_2^4$ and let $I = \delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}) \subset \mathcal{R}$ where $\mathcal{R}$ is a polynomial ring defined in* (8.4) *with $n = m = 4$. The reduced Gröbner basis of I w.r.t. degree-reverse lexicographic ordering consists of the following polynomials*

$$
\begin{array}{llll}
y_2''^2 + y_2'', & x_3' + y_2'' + y_3'' + 1, & y_3' + y_3'', & x_3'' + y_2'' + y_3'', \\
y_3''^2 + y_3'', & x_4' + y_3'', & y_4' + 1, & x_4'' + y_3'', \\
x_1' + y_3'' + 1, & y_1', & x_1'' + y_3'' + 1, & y_1'', \\
x_2' + y_2'', & y_2' + y_2'' + 1, & x_2'' + y_2'' + 1, & y_4'' + 1.
\end{array}
$$

The basis of the $\mathbb{F}_2$-vector space $\mathcal{R}/I$ consists of the following elements

$$1 + I, y_2'' + I, y_3'' + I, y_2'' y_3'' + I.$$

Therefore $\delta_S(\boldsymbol{6}, \boldsymbol{2}) = \dim_{\mathbb{F}_2}(\mathcal{R}/I) = 4$ (see also Table 22).

8.4 Multivariate Ideals and Autocorrelation

In this section we shall describe the relation of autocorrelation of vectorial Boolean functions and ideals in multivariate polynomial ring. The notion of autocorrelation is related to the derivative of Boolean functions. The derivative of an n-variable Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ with respect to $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ is the Boolean function $D_{\boldsymbol{\alpha}} f(x) = f(x + \boldsymbol{\alpha}) + f(x)$.

Definition 8.20 (Autocorrelation)**.** *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. The autocorrelation of a component function $S_{\boldsymbol{b}}$ at $\boldsymbol{\alpha}$ is defined as*

$$\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}) = \sum_{\boldsymbol{v} \in \mathbb{F}_2^n} (-1)^{\widehat{D_{\boldsymbol{\alpha}} S_{\boldsymbol{b}}}(\boldsymbol{v})}.$$

If n and m are relatively small, it is possible to exhaustively compute $\tau_S(\boldsymbol{\alpha}, \boldsymbol{b})$ for all $\boldsymbol{\alpha} \in \mathbb{F}_2^n, \boldsymbol{b} \in \mathbb{F}_2^m$. We refer to $(\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}))_{\boldsymbol{\alpha} \in \mathbb{F}_2^n, \boldsymbol{b} \in \mathbb{F}_2^m}$ as the *autocorrelation table* of S. An example of autocorrelation table is provided in Table 23.

The work in [CKL+19] demonstrates that autocorrelation is closely related to the differential-linear cryptanalysis [LH94]. For instance, the notion of *differential-linear connectivity table* (DLCT) introduced in [BDKW19] to examine the differential-linear property of S-Boxes, coincides with the autocorrelation table. Other related cryptographic weaknesses is the notion of *linear structures* which we will describe in Subsection 8.4.1.

We will now present the construction of an ideal that corresponds to the autocorrelation of a component function of S.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	-8	-8	8	-	-8	-8	8	-	-	-8	-	-	-	8	-
2	16	-8	-	-	-8	-	-	-	-8	-	-	-	-	8	-	-
3	16	8	-	-	-	-	-8	-	-	-	-	-8	-8	-	-	-
4	16	-	-	-8	-	-	-	-8	-8	-	-	-	-8	-	-	16
5	16	-	-	-8	-	-	-	-8	-	-8	-	-	-	8	-	-
6	16	-	-8	-	-	8	-	-	8	-	-8	-8	-	-	-8	-
7	16	-	-	-	-8	-8	-	-	-	-8	8	-	8	-	-8	-
8	16	-	-8	-	-8	-	-	-	-8	-8	-	8	8	-	-	-
9	16	-8	8	-	-	-	-	-	-	-	-	-	-	-8	8	-16
A	16	-8	-	-8	8	-8	-	-	-	-8	8	-	-	-	-	-
B	16	8	-	-8	-	8	-	8	-	-	-8	-8	-8	-8	-	-
C	16	-	-	-	-	-8	8	-	8	-	-8	-	-8	-8	-	-
D	16	-	-	-	-8	-	-8	-	-	8	-	8	-	-	-	-16
E	16	-	8	8	-	-	8	-8	-8	-	-	-8	-	-8	-8	-
F	16	-	-8	-	8	-	-8	-8	-	8	-	-	-	-	-8	-

Table 23: *Autocorrelation table of S defined in Table 20. The entries denoted by '-' are equal to zero.*

Definition 8.21 (Autocorrelation ideal)**.** *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and let $\mathcal{R}$ be a polynomial ring defined in* (8.4)*. For any $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_2^n$ and $\boldsymbol{b} = (b_1, \ldots, b_m) \in \mathbb{F}_2^m$ we define the autocorrelation ideal $\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}) \subseteq \mathcal{R}$ of S with input difference $\boldsymbol{\alpha}$ and output mask $\boldsymbol{b}$ as*

$$\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}) = \langle F'_S, F''_S, \{x'_i + x''_i + \alpha_i : 1 \le i \le n\}, \sum_{j=1}^{m} b_j(y'_j + y''_j) + 1, \mathsf{FP}(\mathcal{R})\}\rangle.$$

where F'_S and F''_S are the set of polynomials that represent S in variables x'_i, y'_j and x''_i, y''_j respectively.

Theorem 8.22. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and let $\mathcal{R}$ be a polynomial ring defined in* (8.4)*. The autocorrelation of a component function $S_{\boldsymbol{b}}$ at $\boldsymbol{\alpha}$ is equal to*

$$\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}) = 2^n - 2 \cdot \dim_{\mathbb{F}_2}(\mathcal{R}/\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})).$$

Proof. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_2^n$ and $\boldsymbol{b} = (b_1, \ldots, b_m) \in \mathbb{F}_2^m$. Suppose that the following system of equations

$$F = F'_S \cup F''_S \cup \{x'_i + x''_i + \alpha_i : 1 \le i \le n\} \cup \{\textstyle\sum_{j=1}^{m} b_j(y'_j + y''_j) + 1\} \cup \mathsf{FP}(\mathcal{R})$$

has a solution in $\mathbb{F}_2^{2(n+m)}$. We first define the following set

$$\mathrm{T}_S(\boldsymbol{\alpha}, \boldsymbol{b}) = \{(\boldsymbol{v} \parallel S(\boldsymbol{v}) \parallel \boldsymbol{v} + \boldsymbol{\alpha} \parallel S(\boldsymbol{v} + \boldsymbol{\alpha})) : \boldsymbol{v} \in \mathbb{F}_2^n, S_{\boldsymbol{b}}(\boldsymbol{v} + \boldsymbol{\alpha}) + S_{\boldsymbol{b}}(\boldsymbol{v}) = 1\}$$

and we will show that $\mathbf{V}_{\mathbb{F}_2}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})) = \mathrm{T}_S(\boldsymbol{\alpha}, \boldsymbol{b})$.

Let $\boldsymbol{u} = (u_1, \ldots, u_{2(n+m)}) \in \mathbf{V}_{\mathbb{F}_2}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}))$. Since $\mathbf{V}_{\mathbb{F}_2}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})) = \mathbf{V}_{\mathbb{F}_2}(F)$, then $f(\boldsymbol{u}) = 0$ for all $f \in F$. Thus, $(u_{n+m+1}, \ldots, u_{2n+m}) = (u_1 + \alpha_1, \ldots, u_n + \alpha_n)$ by the polynomials in $\{x'_i + x''_i + \alpha_i : 1 \le i \le n\}$. Moreover, we have $(u_{n+1}, \ldots, u_{n+m}) = S(u_1, \ldots, u_n)$ and at the same time $(u_{2n+m+1}, \ldots u_{2(n+m)}) = S(u_{n+m+1}, \ldots, u_{2n+m})$ by the definition of F'_S, F''_S respectively such that

$$\sum_{j=1}^{m} b_j(u_{j+n} + u_{j+2n+m}) + 1 = S_{\boldsymbol{b}}(u_1, \ldots, u_n) + S(u_1 + \alpha_1, \ldots, u_n + \alpha_n) + 1 = 0.$$

Therefore, $\boldsymbol{u} \in \mathrm{T}_S(\boldsymbol{\alpha}, \boldsymbol{b})$. Conversely, let $(\boldsymbol{v} \parallel S(\boldsymbol{v}) \parallel \boldsymbol{v} + \boldsymbol{\alpha} \parallel S(\boldsymbol{v} + \boldsymbol{\alpha})) \in \mathrm{T}_S(\boldsymbol{\alpha}, \boldsymbol{b})$ where $\boldsymbol{v} \in \mathbb{F}_2^n$ and $S_{\boldsymbol{b}}(\boldsymbol{v}) + S_{\boldsymbol{b}}(\boldsymbol{v} + \boldsymbol{\alpha}) = 1$. Clearly $f'(\boldsymbol{v} \parallel S(\boldsymbol{v})) = 0$ for all $f' \in F'_S$ and $f''(\boldsymbol{v} + \boldsymbol{\alpha} \parallel S(\boldsymbol{v} + \boldsymbol{\alpha})) = 0$ for all $f'' \in F''_S$. Similarly, $(\boldsymbol{v} \parallel \boldsymbol{v} + \boldsymbol{\alpha})$ is a solution of $x'_i + x''_i + \alpha_i = 0$ for all $1 \leq i \leq n$ and the condition $S_{\boldsymbol{b}}(\boldsymbol{v}) + S_{\boldsymbol{b}}(\boldsymbol{v} + \boldsymbol{\alpha}) = 1$ implies that $(S(\boldsymbol{v}) \parallel S(\boldsymbol{v} + \boldsymbol{\alpha}))$ is a solution for $\sum_{j=1}^{m} b_j(y'_j + y''_j) + 1 = 0$. Thus $(\boldsymbol{v} \parallel S(\boldsymbol{v}) \parallel \boldsymbol{v} + \boldsymbol{\alpha} \parallel S(\boldsymbol{v} + \boldsymbol{\alpha})) \in \mathbf{V}_{\mathbb{F}_2}(F) = \mathbf{V}_{\mathbb{F}_2}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}))$.

Since $\mathsf{FP}(\mathcal{R}) \subseteq \tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})$, by 1) and 2) of Proposition 8.5 the ideal $\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})$ is zero-dimensional and radical. We then have the following relation

$$|\mathbf{V}_{\overline{\mathbb{F}_2}}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}))| = \dim_{\mathbb{F}_2}(\mathcal{R}/\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})) = |\mathbf{V}_{\mathbb{F}_2}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{\beta}))|$$

where the first equality is an implication of Proposition 8.3 and the second equality is an implication of 3) in Proposition 8.5.

By the definition of Hamming weight, clearly $\mathsf{wt}(D_{\boldsymbol{\alpha}} S_{\boldsymbol{b}}) = |\mathbf{V}_{\mathbb{F}_2}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}))|$ and for any n-variable Boolean function f, $\sum_{\boldsymbol{v} \in \mathbb{F}_2^n} (-1)^{\widehat{f(\boldsymbol{v})}} = 2^n - 2 \cdot \mathsf{wt}(f)$ holds. Therefore

$$\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}) = 2^n - 2 \cdot \mathsf{wt}(D_{\boldsymbol{\alpha}} S_{\boldsymbol{b}}) = 2^n - 2 \cdot \dim_{\mathbb{F}_2}(\mathcal{R}/\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}).$$

■

Theorem 8.23. *Given $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$, $\boldsymbol{\alpha} \in \mathbb{F}_2^n$, and $\boldsymbol{b} \in \mathbb{F}_2^m$, Algorithm 8.4 computes the autocorrelation $\tau_S(\boldsymbol{\alpha}, \boldsymbol{b})$.*

Input: A vectorial Boolean function $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$
Input: An input difference $\boldsymbol{\alpha} \in \mathbb{F}_2^n$
Input: An output mask $\boldsymbol{b} \in \mathbb{F}_2^m$
Output: The autocorrelation $\tau_S(\boldsymbol{\alpha}, \boldsymbol{b})$

1 $\mathcal{R} \leftarrow \mathbb{F}_2[x'_1, \ldots, x'_n, y'_1, \ldots, y'_m, x''_1, \ldots, x''_n, y''_1, \ldots, y''_m]$
2 $F'_S \leftarrow$ Algebraic Normal Form of S in $x'_1, \ldots, x'_n, y'_1, \ldots, y'_m$
3 $F''_S \leftarrow$ Algebraic Normal Form of S in $x''_1, \ldots, x''_n, y''_1, \ldots, y''_m$
4 $A \leftarrow \{x'_i + x''_i + \alpha_i : 1 \leq i \leq n\}$
5 $f \leftarrow \sum_{j=1}^{m} b_j(y'_j + y''_j) + 1$
6 $\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}) \leftarrow \langle F'_S, F''_S, A, f, \mathsf{FP}(\mathcal{R}) \rangle$
7 $G \leftarrow$ GröbnerBasis($\tau I_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$)
8 $\mathsf{RM}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})) \leftarrow$ ReducedMonomials(G)
9 **return** $2^n - 2 \cdot |\mathsf{RM}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}))|$

Algorithm 8.4: An algorithm to compute the autocorrelation $\tau_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$ using a Gröbner bases algorithm.

Proof. Termination: Clearly Algorithm 8.4 terminates since the computation of the algebraic normal form of S, Gröbner basis, and the ReducedMonomials eventually terminate.
Correctness: By Proposition 8.2, $\{\mathfrak{m} + \tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}) : \mathfrak{m} \in \mathsf{RM}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}))\}$ is a basis of the $\mathbb{F}_2$-vector space $\mathcal{R}/\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})$ and clearly by Theorem 8.22, $\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}) = 2^n - 2 \cdot |\mathsf{RM}(\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b}))|$. ■

Example 8.24. *Let S be the mapping defined in Table 20. Let $\boldsymbol{\alpha} = \mathtt{5} = (1, 0, 1, 0) \in \mathbb{F}_2^4, \boldsymbol{b} = \mathtt{D} = (1, 0, 1, 1) \in \mathbb{F}_2^4$. The reduced Gröbner basis of the*

ideal $I = \tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})$ *w.r.t. degree-reverse lexicographic ordering consists of the following polynomials*

$$
\begin{array}{lll}
y_1''^2 + y_1'', & x_1' + y_1'' + y_3'' + y_4'' + 1, & y_3' + y_3'', \\
y_1''y_3'' + y_1'', & x_2' + y_3'', & y_4' + y_3'' + y_4'' + 1, \\
y_3''^2 + y_3'', & x_3' + y_1'' + y_3'' + y_4'' + 1, & x_1'' + y_1'' + y_3'' + y_4'', \\
y_1''y_4'', & x_4', & x_2'' + y_3'', \\
y_3''y_4'', & y_1' + y_1'' + y_3'', & x_3'' + y_1'' + y_3'' + y_4'', \\
y_4''^2 + y_4'', & y_2' + y_3'' + y_4'' + 1, & \\
x_4'', & y_2'' + y_4'', &
\end{array}
$$

and the basis of the $\mathbb{F}_2$*-vector space* $\mathcal{R}/I$ *consists of the following elements*

$$y_4'' + I, y_3'' + I, y_1'' + I, 1 + I.$$

Therefore $\tau_S(\mathtt{5}, \mathtt{D}) = 2^4 - 2 \cdot \dim_{\mathbb{F}_2}(\mathcal{R}/I) = 8$ *(see also Table 23).*

8.4.1 Linear Structures and Ideal Membership Problem

One particular application of autocorrelation is to determine whether a vectorial Boolean function has a linear structure. The vector $\boldsymbol{\alpha}$ is a linear structure of an n-variable Boolean function f if $D_{\boldsymbol{\alpha}} f$ is a constant function. If the constant function $D_{\boldsymbol{\alpha}} f$ is equal to $c \in \mathbb{F}_2$, we say that $\boldsymbol{\alpha}$ is a c-linear structure of f. A Boolean function f is said to have a linear structure if there exists a nonzero $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ such that $D_{\boldsymbol{\alpha}} f$ is a constant function [Eve87, Lai94]. This definition was generalized for the case of vectorial Boolean functions in [MS89b, pg. 4].

Definition 8.25. *A vectorial Boolean function* $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ *is said to have a linear structure if one of its nontrivial component function has a linear structure, i.e., there exists a nonzero* $\boldsymbol{b} \in \mathbb{F}_2^m$ *and a nonzero* $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ *such that* $S_{\boldsymbol{b}}(x + \boldsymbol{\alpha}) + S_{\boldsymbol{b}}(x)$ *is a constant function.*

Clearly, S has a linear structure if and only if there exists a nonzero $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ and a nonzero $\boldsymbol{b} \in \mathbb{F}_2^m$ such that $\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}) = \pm 2^n$. The magnitude of $\tau(\boldsymbol{\alpha}, \boldsymbol{b})$ tells about the constant function $D_{\boldsymbol{\alpha}} S_{\boldsymbol{b}}$, i.e., it is a 0-linear structure (resp. 1-linear structure) if and only if $\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}) = 2^n$ (resp. $\tau_S(\boldsymbol{\alpha}, \boldsymbol{b}) = -2^n$). From the autocorrelation table in Table 23, one can see that the function defined in Table 20 has a linear structure (see the entries in the column `F`). The following corollary is an immediate consequences of Theorem 8.22.

Corollary 8.26. *Let* $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$*. A vector* $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ *is a linear structure of the component function* $S_{\boldsymbol{b}}$ *if and only if* $\dim_{\mathbb{F}_2}(\mathcal{R}/\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})) \in \{0, 2^n\}$.

We will now discuss other possible means to characterize whether a vectorial Boolean functions has a linear structure. We first recall the following result that describes the relation of a linear structure, when viewed as an input difference to S, and the set of all its possible output differences.

Proposition 8.27. *Let* $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$*. The vector* $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ *is a* c*-linear structure of the component function* $S_{\boldsymbol{b}}$ *if and only if* $\boldsymbol{b} \cdot \boldsymbol{\beta} = c$ *for all* $\boldsymbol{\beta} \in \mathbb{F}_2^m$ *such that* $\delta_S(\boldsymbol{\alpha}, \boldsymbol{\beta}) > 0$.

Proof. See the proof of Theorem 7 in [MT14, pg. 17]. ■

Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_2^n$, $\boldsymbol{b} = (b_1, \ldots, b_m) \in \mathbb{F}_2^m$. In the construction of autocorrelation ideals $\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})$, we look at the implication of the derivative of $S_{\boldsymbol{b}}$ w.r.t. $\boldsymbol{\alpha}$. However, the construction of this ideal is too restrictive when we want to characterize $\boldsymbol{\alpha}$ as a linear structure of some component functions of S because in this case $\boldsymbol{\alpha}$ is the only relevant parameter.

Recall that we may view $\boldsymbol{\alpha}$ as an input difference to S. A set of polynomials in $\mathcal{R}$ that describes the application of $\boldsymbol{\alpha}$ to two different inputs of S is given by

$$F'_S \cup F''_S \cup \{x'_i + x''_i + \alpha_i : 1 \leq i \leq n\}$$

where F'_S and F''_S are the set of polynomials that represent S in variables x'_i, y'_j and x''_i, y''_j respectively, $1 \leq i \leq n, 1 \leq j \leq m$. If $\boldsymbol{\alpha}$ is a linear structure to some component functions of S, then the ideal

$$\langle F'_S \cup F''_S \cup \{x'_i + x''_i + \alpha_i : 1 \leq i \leq n\}\rangle$$

contains all polynomials that are implied by having $\boldsymbol{\alpha}$ as an input difference to S. One possible element of $\langle F'_S \cup F''_S \cup \{x'_i + x''_i + \alpha_i : 1 \leq i \leq n\}\rangle$ is the polynomial that represents Proposition 8.27, i.e.

$$\left(\sum_{j=1}^{m} b_j(y'_j + y''_j)\right) + c = \left(\sum_{j=1}^{m} b_j y'_j + b_j y''_j\right) + c. \tag{8.6}$$

Conjecture 8.28. *Let $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and let $\mathcal{R}$ be a polynomial ring defined in* (8.4)*. A vector $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ is a c-linear structure of the component function $S_{\boldsymbol{b}}$, $\boldsymbol{b} = (b_1, \ldots, b_m) \in \mathbb{F}_2^m$, if and only if*

$$\left(\sum_{j=1}^{m} b_j y'_j + b_j y''_j\right) + c \in \langle F'_S, F''_S, \{x'_i + x''_i + \alpha_i : 1 \leq i \leq n\}\rangle$$

where F'_S and F''_S are the set of polynomials that represent S in variables x'_i, y'_j and x''_i, y''_j respectively.

Note that by computing a Gröbner bases of the ideal $I = \langle F'_S, F''_S, \{x'_i + x''_i + \alpha_i : 1 \leq i \leq n\}\rangle$, we can determine whether the polynomial of the form (8.6) is in I (see Corollary 3.38). In the following, we provide an example that demonstrate Conjecture 8.28.

Example 8.29. *Let S be the mapping defined in Table 20 and let $\boldsymbol{\alpha} = 4 = (0, 0, 1, 0) \in \mathbb{F}_2^4$. We construct the polynomial ring $\mathcal{R}$ as in* (8.4) *with $n = m = 4$ and use lexicographic monomial ordering in $\mathcal{R}$. The reduced Gröbner basis G of*

$I = \langle F'_S, F''_S, \{x'_i + x''_i + \alpha_i : 1 \le i \le 4\}\rangle$ *consists of the following polynomials*

$$
\begin{aligned}
g_1 &= x'_1 + y''_1y''_2y''_4 + y''_1y''_3y''_4 + y''_1y''_3 + y''_1y''_4 + y''_2y''_4 + y''_2 + y''_3 + y''_4,\\
g_2 &= x'_2 + y''_1y''_2y''_3 + y''_1y''_2 + y''_1y''_3y''_4 + y''_2 + y''_3y''_4 + y''_3 + y''_4,\\
g_3 &= x'_3 + y''_1y''_2y''_4 + y''_1y''_2 + y''_1y''_3y''_4 + y''_1y''_4 + y''_1 + y''_2y''_4 + y''_3 + 1,\\
g_4 &= x'_4 + y''_1y''_2y''_3 + y''_1y''_3 + y''_1 + y''_2y''_3y''_4 + y''_2y''_3 + y''_2y''_4 + y''_2 + y''_3y''_4,\\
g_5 &= y'_1 + y''_1y''_2y''_3 + y''_1y''_2 + y''_1y''_3y''_4 + y''_1y''_3 + y''_2y''_3 + y''_2 + y''_3y''_4,\\
g_6 &= y'_2 + y''_1y''_2y''_3 + y''_1y''_3 + y''_2y''_3y''_4 + y''_2y''_4 + y''_3y''_4 + y''_3 + y''_4 + 1,\\
g_7 &= y'_3 + y''_1y''_3 + y''_2y''_3 + y''_2 + y''_3 + 1,\\
g_8 &= y'_4 + y''_1y''_2 + y''_1y''_3y''_4 + y''_1y''_3 + y''_1 + y''_2y''_3y''_4 + y''_2y''_4 + y''_2 + y''_3,\\
g_9 &= x''_1 + y''_1y''_2y''_4 + y''_1y''_3y''_4 + y''_1y''_3 + y''_1y''_4 + y''_2y''_4 + y''_2 + y''_3 + y''_4,\\
g_{10} &= x''_2 + y''_1y''_2y''_3 + y''_1y''_2 + y''_1y''_3y''_4 + y''_2 + y''_3y''_4 + y''_3 + y''_4,\\
g_{11} &= x''_3 + y''_1y''_2y''_4 + y''_1y''_2 + y''_1y''_3y''_4 + y''_1y''_4 + y''_1 + y''_2y''_4 + y''_3,\\
g_{12} &= x''_4 + y''_1y''_2y''_3 + y''_1y''_3 + y''_1 + y''_2y''_3y''_4 + y''_2y''_3 + y''_2y''_4 + y''_2 + y''_3y''_4,\\
g_{13} &= y''^2_1 + y''_1,\\
g_{14} &= y''^2_2 + y''_2,\\
g_{15} &= y''^2_3 + y''_3,\\
g_{16} &= y''^2_4 + y''_4.
\end{aligned}
$$

Let $\boldsymbol{b} = F = (1,1,1,1) \in \mathbb{F}_2^4$ *and let* $f = \sum_{i=1}^4 b_i y'_i + b_i y''_i + 0$. *We see that* $f \in I = \langle G\rangle$ *because* $f = g_5 + g_6 + g_7 + g_8$. *Therefore* $\boldsymbol{4}$ *is a* 0*-linear structure of the component function* $S_{\boldsymbol{b}}$ *(see also that* $\tau_S(\boldsymbol{4}, F) = 16$ *in Table 23).*

Note that if the ring $\mathcal{R}$ used a monomial ordering that respects the degree of monomials (e.g., degree-lexicographic or degree-reverse lexicographic ordering), it is possible that a polynomial of the form (8.6) is an element of the Gröbner basis. For instance, if we use the ring

$$\mathcal{R} = \mathbb{F}_2[x'_1, x'_2, x'_3, x'_4, x''_1, x''_2, x''_3, x''_4, y'_1, y'_2, y'_3, y'_4, y''_1, y''_2, y''_3, y''_4]$$

together with degree-lexicographic ordering, the polynomials f will appear in a Gröbner bases G of I. This, however, is not the case if we use different ordering of variables such as $\mathcal{R} = \mathbb{F}_2[x'_1, x'_2, x'_3, x'_4, y'_1, y'_2, y'_3, y'_4, x''_1, x''_2, x''_3, x''_4, y''_1, y''_2, y''_3, y''_4]$.

More generally, assuming Conjecture 8.28 is true, if $\mathcal{R}$ uses a monomial ordering that respects the degree of monomials and S has a linear structure, then there exists a polynomial of degree 1 in a Gröbner basis of $\langle F'_S, F''_S, \{x'_i + x''_i + \alpha_i : 1 \le i \le n\}\rangle$.

A Solutions to Type V and VI MQ Challenge

Type V - (n, m) = (24, 16) - Seed 1
(46, 0C, 42, 10, 26, 4F, 43, 31, C4, 9D, 34, 55, 3A, C9, 39, 3F, D4, 3A, 88, E5, 05, 17, 2D, 4B)

Type V - (n, m) = (25, 17) - Seed 0
(49, BB, 5E, C9, E, BF, 1B, FE, 9E, F6, C8, DC, 4C, 3, C9, AA, 53, 20, 34, 1A, 16, 62, 94, 45, 8C)

Type V - (n, m) = (27, 18) - Seed 0
(21, 21, 9E, 93, 6A, 19, 3E, CE, A2, 1A, 5D, 19, D3, D7, 16, 5, 9F, E1, 0, DA, 92, 55, EB, 19, 89, F0, 1)

Type V - (n, m) = (28, 19) - Seed 0
(F0, E2, 62, D7, 7F, A7, B3, 72, 51, 91, AD, 89, 57, F3, E6, 42, 4C, 63, C2, 16, FA, 6B, 54, 9B, 10, 62, C3, 34)

Type VI - (n, m) = (24, 16) - Seed 1
(30, 9, 25, 25, 23, 22, 25, 18, 4, 25, 23, 30, 26, 13, 19, 14, 30, 29, 6, 0, 3, 14, 23, 23)

Type VI - (n, m) = (25, 17) - Seed 0
(8, 1, 4, 15, 3, 14, 18, 16, 28, 2, 29, 24, 11, 17, 18, 8, 26, 27, 12, 23, 26, 18, 11, 26, 6)

Type VI - (n, m) = (27, 18) - Seed 0
(19, 19, 20, 30, 13, 12, 6, 11, 3, 16, 12, 14, 29, 29, 26, 0, 13, 11, 20, 17, 1, 11, 30, 11, 14, 2, 24)

Type VI - (n, m) = (28, 19) - Seed 0
(12, 26, 6, 4, 9, 24, 21, 8, 0, 19, 23, 28, 11, 24, 22, 5, 9, 24, 24, 13, 0, 19, 25, 25, 26, 10, 27, 16)

Type VI - (n, m) = (30, 20) - Seed 0
(11, 4, 28, 8, 10, 20, 26, 20, 30, 28, 0, 13, 26, 2, 22, 9, 8, 26, 7, 6, 9, 26, 24, 15, 22, 18, 12, 3, 11, 13)

B CPU Time and Memory usage of M4GB

$\mathbb{F}_{2^8}$, $m-n=1$			
n	m	CPU time	Memory
15	16	2121.83 s	1149 MB
16	17	18454.55 s	4557 MB
17	18	96000.30 s	14935 MB
18	19	840535.63 s	57000 MB
$\mathbb{F}_{2^8}$, $m-n=2$			
n	m	CPU time	Memory
14	16	81.73 s	108 MB
15	17	489.34 s	395 MB
16	18	2593.36 s	1169 MB
17	19	21461.78 s	4696 MB
18	20	106369.83 s	14880 MB
19	21	930243.86 s	55410 MB
$\mathbb{F}_{2^8}$, $m-n=3$			
n	m	CPU time	Memory
13	16	5.85 s	21 MB
14	17	35.56 s	66 MB
15	18	132.68 s	167 MB
16	19	1148.80 s	576 MB
17	20	4929.98 s	1723 MB
18	21	26096.75 s	5334 MB
19	22	207408.54 s	19835 MB
20	23	1115986.42 s	74004 MB

Table 24: *The CPU time and memory usage of* M4GB *to solve system of polynomials over* $\mathbb{F}_{2^8}$.

$\mathbb{F}_{31}, m-n=1$			
n	m	CPU time	Memory
15	16	2157.01 s	1149 MB
16	17	18592.99 s	4559 MB
17	18	119299.90 s	14868 MB
18	19	847613.39 s	57011 MB

$\mathbb{F}_{31}$, $m-n=2$			
n	m	CPU time	Memory
14	16	81.47 s	109 MB
15	17	491.49 s	398 MB
16	18	2620.79 s	1168 MB
17	19	21632.05 s	4657 MB
18	20	107099.96 s	14868 MB
19	21	941960.45 s	55516 MB

$\mathbb{F}_{31}$, $m-n=3$			
n	m	CPU time	Memory
13	16	6.29 s	21 MB
14	17	36.69 s	66 MB
15	18	134.21 s	164 MB
16	19	1157.86 s	580 MB
17	20	4961.64 s	1733 MB
18	21	26277.35 s	5343 MB
19	22	208961.14 s	19842 MB
20	23	1124110.35 s	74265 MB
21	24	1729007.02 s	126538 MB

$\mathbb{F}_{31}$, $m-n=4$			
n	m	CPU Time	Memory
12	16	0.83 s	10 MB
13	17	4.47 s	18 MB
14	18	10.13 s	31 MB
15	19	80.41 s	100 MB
16	20	284.43 s	275 MB
17	21	1640.92 s	796 MB
18	22	11400.29s	2749 MB
19	23	51281.00 s	8438 MB
20	24	323866.14 s	28865 MB
21	25	2114895.49 s	99646 MB

$\mathbb{F}_{31}$, $m-n=5$			
n	m	CPU Time	Memory
11	16	0.44 s	10 MB
12	17	0.80 s	10 MB
13	18	1.55 s	10 MB
14	19	7.54 s	25 MB
15	20	30.13 s	52 MB
16	21	181.32 s	154 MB
17	22	719.10 s	460 MB
18	23	3538.91 s	1368 MB
19	24	25530.18 s	4240 MB
20	25	111324.47 s	14035 MB
21	26	678252.18 s	46196 MB

$\mathbb{F}_{31}$, $m-n=6$			
n	m	CPU Time	Memory
10	16	0.12 s	9 MB
11	17	0.39 s	10 MB
12	18	0.77 s	10 MB
13	19	1.61 s	10 MB
14	20	6.40 s	20 MB
15	21	14.16 s	37 MB
16	22	72.64 s	83 MB
17	23	417.50 s	268 MB
18	24	1604.10 s	733 MB
19	25	8600.97 s	2468 MB
20	26	59280.30 s	7332 MB
21	27	262183.34 s	24526 MB
22	28	1465417.71 s	78768 MB

$\mathbb{F}_{31}$, $m-n=7$			
n	m	CPU Time	Memory
9	16	0.07 s	8 MB
10	17	0.12 s	9 MB
11	18	0.16 s	9 MB
12	19	0.66 s	9 MB
13	20	1.27 s	10 MB
14	21	2.29 s	11 MB
15	22	12.90 s	30 MB
16	23	42.25 s	64 MB
17	24	173.77 s	164 MB
18	25	1016.54 s	468 MB
19	26	3641.07 s	1301 MB
20	27	21341.13 s	4274 MB
21	28	139796.22 s	12595 MB
22	29	578488.31 s	40602 MB

$\mathbb{F}_{31}$, $m-n=8$			
n	m	CPU Time	Memory
8	16	0.05 s	7 MB
9	17	0.10 s	8 MB
10	18	0.13 s	9 MB
11	19	0.18 s	9 MB
12	20	0.28 s	8 MB
13	21	1.07 s	9 MB
14	22	1.97 s	11 MB
15	23	7.44 s	17 MB
16	24	25.80 s	47 MB
17	25	100.40 s	108 MB
18	26	440.25 s	288 MB
19	27	2414.14 s	807 MB
20	28	9464.87 s	2311 MB
21	29	52474.98 s	7678 MB
22	30	320730.95 s	22185 MB
23	31	1345880.83 s	70750 MB

Table 25: *The CPU time and memory usage of* M4GB *to solve system of polynomials over* $\mathbb{F}_{31}$.

References

[AC09] Martin R. Albrecht and Carlos Cid, *Algebraic Techniques in Differential Cryptanalysis*, Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers (Orr Dunkelman, ed.), Lecture Notes in Computer Science, vol. 5665, Springer, 2009, pp. 193–208.

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret, *Algebraic algorithms for LWE problems*, ACM Commun. Comput. Algebra **49** (2015), no. 2, 62.

[ACFP12] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret, *On the relation between the MXL family of algorithms and Gröbner basis algorithms*, J. Symb. Comput. **47** (2012), no. 8, 926–941.

[ADH+19] Martin R. Albrecht, Lèo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens, *The General Sieve Kernel (G6K)*, Available at `https://github.com/fplll/g6k`, 2019.

[AFI+04] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita, *Comparison Between XL and Gröbner Basis Algorithms*, Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings (Pil Joong Lee, ed.), Lecture Notes in Computer Science, vol. 3329, Springer, 2004, pp. 338–353.

[AG11] Sanjeev Arora and Rong Ge, *New Algorithms for Learning in Presence of Errors*, Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I (Luca Aceto, Monika Henzinger, and Jirí Sgall, eds.), Lecture Notes in Computer Science, vol. 6755, Springer, 2011, pp. 403–415.

[AIK+00] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita, *Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis*, in Stinson and Tavares [ST01], pp. 39–56.

[AK03] Frederik Armknecht and Matthias Krause, *Algebraic Attacks on Combiners with Memory*, in Boneh [Bon03], pp. 162–175.

[AL94] William W. Adams and Philippe Loustaunau, *An Introduction to Grobner Bases*, American Mathematical Society (AMS), 1994.

[Alb07] Martin Albrecht, *Algebraic Attacks on the Courtois Toy Cipher*, Master's thesis, Universität Bremen, 2007.

[AP10] Martin Albrecht and John Perry, *F4/5*, 2010.

[AST24] Thomas Aulbach, Simona Samardjiska, and Monika Trimoska, *Practical Key-Recovery Attack on MQ-Sign and More*, Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Oxford, UK, June 12-14, 2024, Proceedings, Part II (Markku-Juhani O. Saarinen and Daniel Smith-Tone, eds.), Lecture Notes in Computer Science, vol. 14772, Springer, 2024, pp. 168–185.

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen, *Serpent: A New Block Cipher Proposal*, Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings (Serge Vaudenay, ed.), Lecture Notes in Computer Science, vol. 1372, Springer, 1998, pp. 222–238.

[Bar04] Magali Bardet, *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie*, Theses, Université Pierre et Marie Curie - Paris VI, December 2004.

[Bar07] Gregory Bard, *Algorithms for Solving Linear and Polynomial Systems of Equations over Finite Fields with Applications to Cryptanalysis*, Ph.D. thesis, Department of Mathematics, University of Maryland, 2007.

[Bar09] Gregory Bard, *Algebraic Cryptanalysis*, Springer, 2009.

[Bay82] David Allen Bayer, *The Division Algorithm and the Hilbert Scheme*, Ph.D. thesis, Harvard University, Cambridge, MA, USA, 1982, AAI8222588.

[BBB+20] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich, *An Algebraic Attack on Rank Metric Code-Based Cryptosystems*, Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III (Anne Canteaut and Yuval Ishai, eds.), Lecture Notes in Computer Science, vol. 12107, Springer, 2020, pp. 64–93.

[BBD09] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen (eds.), *Post-Quantum Cryptography*, Springer Berlin, Heidelberg, 2009.

[BBM23] Stefano Barbero, Emanuele Bellini, and Rusydi H. Makarim, *Rotational analysis of ChaCha permutation*, Adv. Math. Commun. **17** (2023), no. 6, 1422–1439.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir, *Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials*, in Stern [Ste99], pp. 12–23.

[BC03] Alex Biryukov and Christophe De Cannière, *Block Ciphers and Systems of Quadratic Equations*, Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers (Thomas Johansson, ed.), Lecture Notes in Computer Science, vol. 2887, Springer, 2003, pp. 274–289.

[BCC+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang, *Fast Exhaustive Search for Polynomial Systems in* $\mathbb{F}_2$, Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings (Stefan Mangard and François-Xavier Standaert, eds.), Lecture Notes in Computer Science, vol. 6225, Springer, 2010, pp. 203–218.

[BCC+13] Charles Bouillaguet, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang, *Fast Exhaustive Search for Quadratic Systems in $$\mathbb {F}_{2}$$ on FPGAs*, Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers (Tanja Lange, Kristin E. Lauter, and Petr Lisonek, eds.), Lecture Notes in Computer Science, vol. 8282, Springer, 2013, pp. 205–222.

[BCC+23] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer, *MAYO Specification*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[BCD+23] Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang, *UOV: Unbalanced Oil and Vinegar*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[BCJ07] Gregory V. Bard, Nicolas Courtois, and Chris Jefferson, *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers*, IACR Cryptology ePrint Archive **2007** (2007), 24.

[BCM+19] Emanuele Bellini, Florian Caullery, Rusydi H. Makarim, Marc Manzano, Chiara Marcolla, and Víctor Mateu, *Advances and Challenges of Rank Metric Cryptography Implementations*, 37th IEEE International Conference on Computer Design, ICCD 2019, Abu Dhabi, United Arab Emirates, November 17-20, 2019, IEEE, 2019, pp. 325–328.

[BCP97] Wieb Bosma, John J. Cannon, and Catherine Playoust, *The Magma Algebra System I: The User Language*, J. Symb. Comput. **24** (1997), no. 3/4, 235–265.

[BD07] Eli Biham and Orr Dunkelman, *Differential Cryptanalysis in Stream Ciphers*, IACR Cryptology ePrint Archive **2007** (2007), 218.

[BD09a] Michael Brickenstein and Alexander Dreyer, *PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials*, J. Symb. Comput. **44** (2009), no. 9, 1326–1345.

[BD09b] Michael Brickenstein and Alexander Dreyer, *PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials*, Journal of Symbolic Computation **44** (2009), no. 9, 1326 – 1345, Effective Methods in Algebraic Geometry.

[BDDL16] Xavier Bultel, Jannik Dreier, Jean-Guillaume Dumas, and Pascal Lafourcade, *Physical Zero-Knowledge Proofs for Akari, Takuzu, Kakuro and KenKen*, 8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy (Erik D. Demaine and Fabrizio Grandoni, eds.), LIPIcs, vol. 49, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 8:1–8:20.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé, *CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM*, 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018, IEEE, 2018, pp. 353–367.

[BDKW19] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman, *DLCT: A New Tool for Differential-Linear Cryptanalysis*, Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I (Yuval Ishai and Vincent Rijmen, eds.), Lecture Notes in Computer Science, vol. 11476, Springer, 2019, pp. 313–342.

[BDZ04] Feng Bao, Robert H. Deng, and Jianying Zhou (eds.), *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography, Singapore, March 1-4, 2004*, Lecture Notes in Computer Science, vol. 2947, Springer, 2004.

[BERW08] Andrey Bogdanov, Thomas Eisenbarth, Andy Rupp, and Christopher Wolf, *Time-Area Optimized Public-Key Engines: MQ-Cryptosystems as Replacement for Elliptic Curves?*, Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings (Elisabeth Oswald and Pankaj Rohatgi, eds.), Lecture Notes in Computer Science, vol. 5154, Springer, 2008, pp. 45–61.

[Beu22] Ward Beullens, *Breaking Rainbow Takes a Weekend on a Laptop*, Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II (Yevgeniy Dodis and Thomas Shrimpton, eds.), Lecture Notes in Computer Science, vol. 13508, Springer, 2022, pp. 464–479.

[Beu24] Ward Beullens, *Improved Cryptanalysis of SNOVA*, IACR Cryptol. ePrint Arch. (2024), 1297.

[BFP09] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret, *Hybrid approach for solving multivariate systems over finite fields*, J. Mathematical Cryptology **3** (2009), no. 3, 177–197.

[BFP12] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret, *Solving polynomial systems over finite fields: improved analysis of the hybrid approach*, in van der Hoeven and van Hoeij [vdHvH12], pp. 67–74.

[BFS04] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy, *On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equation*, Proceedings of the International Conference on Polynomial System Solving, 2004, pp. 71–75.

[BFSS13] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer, *On the complexity of solving quadratic Boolean systems*, J. Complex. **29** (2013), no. 1, 53–75.

[BFSY05] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Bo-Yin Yang, *Asymptotic Behaviour of the Degree of Regularity of Semi-Regular Polynomial Systems*, Eighth International Symposium on Effective Methods in Algebraic Geometry, Porto Conte Alghero, Sardinia (Italy), 2005, pp. 1–14.

[BGG+23a] Emanuele Bellini, David Gérault, Juan Grados, Yun Ju Huang, Rusydi H. Makarim, Mohamed Rachidi, and Sharwan K. Tiwari, *CLAASP: A Cryptographic Library for the Automated Analysis of Symmetric Primitives*, Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers (Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, eds.), Lecture Notes in Computer Science, vol. 14201, Springer, 2023, pp. 387–408.

[BGG+23b] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin, *Boosting Differential-Linear Cryptanalysis of ChaCha7 with MILP*, IACR Trans. Symmetric Cryptol. **2023** (2023), no. 2, 189–223.

[BGG+23c] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin, *Fully Automated Differential-Linear Attacks Against ARX Ciphers*, Topics in Cryptology - CT-RSA 2023 - Cryptographers' Track at the RSA Conference 2023, San Francisco, CA, USA, April 24-27, 2023, Proceedings (Mike Rosulek, ed.), Lecture Notes in Computer Science, vol. 13871, Springer, 2023, pp. 252–276.

[BGMS24] Emanuele Bellini, Juan Grados, Rusydi H. Makarim, and Carlo Sanna, *Finding differential trails on ChaCha by means of state functions*, Int. J. Appl. Cryptogr. **4** (2024), no. 3/4, 156–175.

[Bie17] Armin Biere, *CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017*, Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions (Tomáš Balyo, Marijn Heule, and Matti Järvisalo, eds.), Department of Computer Science Series of Publications B, vol. B-2017-1, University of Helsinki, 2017, pp. 14–15.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe, *PRESENT: An Ultra-Lightweight Block Cipher*, Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007,

Proceedings (Pascal Paillier and Ingrid Verbauwhede, eds.), Lecture Notes in Computer Science, vol. 4727, Springer, 2007, pp. 450–466.

[BKPV23] Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier Verbel, *Biscuit: Shorter MPC-based Signature from PoSSo*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[BKS09] Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe, *Bivium as a Mixed-Integer Linear Programming Problem*, Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Proceedings (Matthew G. Parker, ed.), Lecture Notes in Computer Science, vol. 5921, Springer, 2009, pp. 133–152.

[BKW19] Andreas Björklund, Petteri Kaski, and Ryan Williams, *Solving Systems of Polynomial Equations over GF(2) by a Parity-Counting Self-Reduction*, 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece (Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, eds.), LIPIcs, vol. 132, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 26:1–26:13.

[Blu01] BluetoothSIG, *Specification of the Bluetooth System*, 1.1 ed., February 2001.

[BM22] Emanuele Bellini and Rusydi H. Makarim, *Functional Cryptanalysis: Application to reduced-round Xoodoo*, IACR Cryptol. ePrint Arch. (2022), 134.

[BMS16] Emanuele Bellini, Teo Mora, and Massimiliano Sala, *An algorithmic approach using multivariate polynomials for the nonlinearity of Boolean functions*, CoRR **abs/1610.05932** (2016).

[BMSV22] Emanuele Bellini, Rusydi H. Makarim, Carlo Sanna, and Javier A. Verbel, *An Estimator for the Hardness of the MQ Problem*, Progress in Cryptology - AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18-20, 2022, Proceedings (Lejla Batina and Joan Daemen, eds.), Lecture Notes in Computer Science, vol. 13503, Springer Nature Switzerland, 2022, pp. 323–347.

[Bon03] Dan Boneh (ed.), *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, Lecture Notes in Computer Science, vol. 2729, Springer, 2003.

[BP09] Stanislav Bulygin and Ruud Pellikaan, *Bounded distance decoding of linear error-correcting codes with Gröbner bases*, J. Symb. Comput. **44** (2009), no. 12, 1626–1643.

[BP17] Ward Beullens and Bart Preneel, *Field Lifting for Smaller UOV Public Keys*, Progress in Cryptology - INDOCRYPT 2017 - 18th International Conference on Cryptology in India, Chennai, India, December 10-13, 2017, Proceedings (Arpita Patra and Nigel P. Smart,

eds.), Lecture Notes in Computer Science, vol. 10698, Springer, 2017, pp. 227–246.

[BPM+21] Emanuele Bellini, Alessandro De Piccoli, Rusydi H. Makarim, Sergio Polese, Lorenzo Riva, and Andrea Visconti, *New Records of Preimage Search of Reduced SHA-1 Using SAT Solvers*, Proceedings of the Seventh International Conference on Mathematics and Computing - ICMC 2021, Shibpur, India, (Debasis Giri, Kim-Kwang Raymond Choo, Saminathan Ponnusamy, Weizhi Meng, Sedat Akleylek, and Santi Prasad Maity, eds.), Advances in Intelligent Systems and Computing, vol. 1412, Springer, 2021, pp. 141–151.

[BPSV17] Ward Beullens, Bart Preenel, Alan Szepieniec, and Frederik Vercauteren, *LUOV, Signature Scheme Proposal for NIST PQC Project*, Tech. report, National Institute for Standards and Technology (NIST), 2017.

[BR00] Paulo Barreto and Vincent Rijmen, *The Khazad Legacy-Level Block Cipher*, Submission to the NESSIE Project, 2000.

[Bri10] Michael Brickenstein, *Boolean Gröbner bases - Theory, Algorithms, and Applications*, Ph.D. thesis, Technische Universität Kaiserslautern, 2010.

[BS90] Eli Biham and Adi Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings (Alfred Menezes and Scott A. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer, 1990, pp. 2–21.

[BS23] Charles Bouillaguet and Julia Sauvage, *High-Performance Xbred*, September 2023.

[BSS14] Emanuele Bellini, I. Simonetti, and Massimiliano Sala, *Nonlinearity of Boolean functions: an algorithmic approach based on multivariate polynomials*, CoRR **abs/1404.2741** (2014).

[Buc65] B. Buchberger, *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, Ph.D. thesis, University of Innsbruck, 1965.

[Buc79] Bruno Buchberger, *A criterion for detecting unnecessary reductions in the construction of Groebner bases*, Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings (Edward W. Ng, ed.), Lecture Notes in Computer Science, vol. 72, Springer, 1979, pp. 3–21.

[Buc06] Bruno 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** (2006), no. 3, 475 – 511.

[BW93] Thomas Becker and Volker Weispfenning, *Gröbner Bases - A Computational Approach to Commutative Algebra*, Graduate Texts in Mathematics, vol. 141, Springer New York, 1993.

[Can06] Christophe De Cannière, *Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles*, Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings (Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, eds.), Lecture Notes in Computer Science, vol. 4176, Springer, 2006, pp. 171–186.

[Car10] Claude Carlet, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, ch. Boolean Functions for Cryptography and Error Correcting Codes, pp. 257–397, Cambridge University Press, 2010.

[CB07] Nicolas Courtois and Gregory V. Bard, *Algebraic Cryptanalysis of the Data Encryption Standard*, Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings (Steven D. Galbraith, ed.), Lecture Notes in Computer Science, vol. 4887, Springer, 2007, pp. 152–169.

[CBD+09] Crystal Clough, John Baena, Jintai Ding, Bo-Yin Yang, and Ming-Shing Chen, *Square, a New Multivariate Encryption Scheme*, Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings (Marc Fischlin, ed.), Lecture Notes in Computer Science, vol. 5473, Springer, 2009, pp. 252–264.

[CCNY12] Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang, *Solving Quadratic Equations with XL on Parallel Architectures*, Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings (Emmanuel Prouff and Patrick Schaumont, eds.), Lecture Notes in Computer Science, vol. 7428, Springer, 2012, pp. 356–373.

[CGMT02] Nicolas T. Courtois, Louis Goubin, Willi Meier, and Jean-Daniel Tacier, *Solving Underdefined Systems of Multivariate Quadratic Equations*, Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings (David Naccache and Pascal Paillier, eds.), Lecture Notes in Computer Science, vol. 2274, Springer, 2002, pp. 211–227.

[Che16] Chen-Mou Cheng, *Solving the "CS" Challenge with MiniGB*, Seminar Abstract, March 2016.

[CHR+17] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe, *MQDSS Specifications*, Tech. report, National Institute for Standards and Technology (NIST), 2017.

[CKL+19] Anne Canteaut, Lukas Kölsch, Chao Li, Chunlei Li, Kangquan Li, Longjiang Qu, and Friedrich Wiemer, *On the Differential-Linear Connectivity Table of Vectorial Boolean Functions*, CoRR **abs/1908.07445** (2019).

[CKM97] Stéphane Collart, Michael Kalkbrener, and Daniel Mall, *Converting Bases with the Gröbner Walk*, J. Symb. Comput. **24** (1997), no. 3/4, 465–469.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding (Bart Preneel, ed.), Lecture Notes in Computer Science, vol. 1807, Springer, 2000, pp. 392–407.

[CLO15] David A. Cox, John Little, and Donal O'Shea, *Ideals, Varieties, and Algorithms - An Introduction to Computational Algebraic Geometry and Commutative Algebra*, 4th ed., Undergraduate Texts in Mathematics, Springer International Publishing, 2015.

[CM03] Nicolas Courtois and Willi Meier, *Algebraic Attacks on Stream Ciphers with Linear Feedback*, Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings (Eli Biham, ed.), Lecture Notes in Computer Science, vol. 2656, Springer, 2003, pp. 345–359.

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

[CMR06] Carlos Cid, Sean Murphy, and Matthew Robshaw, *Algebraic Aspects of the Advanced Encryption Standard*, Springer, 2006.

[Coo71] Stephen A. Cook, *The Complexity of Theorem-Proving Procedures*, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA (Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, eds.), ACM, 1971, pp. 151–158.

[Cop94] Don Coppersmith, *Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm*, Mathematics of Computation **62** (1994), 333–350.

[Cou03] Nicolas Courtois, *Fast Algebraic Attacks on Stream Ciphers with Linear Feedback*, in Boneh [Bon03], pp. 176–194.

[Cou04] Nicolas Courtois, *Algebraic Attacks over GF(2^k), Application to HFE Challenge 2 and Sflash-v2*, in Bao et al. [BDZ04], pp. 201–217.

[CP02] Nicolas Courtois and Josef Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings (Yuliang Zheng, ed.), Lecture Notes in Computer Science, vol. 2501, Springer, 2002, pp. 267–287.

[CP03] Nicolas Courtois and Jacques Patarin, *About the XL Algorithm over GF(2)*, Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings (Marc Joye, ed.), Lecture Notes in Computer Science, vol. 2612, Springer, 2003, pp. 141–157.

[CS09] Thomas W. Cusick and Pantelimon Stănică, *Cryptographic Boolean Functions and Applications*, Elsevier Inc, 2009.

[CV05] Anne Canteaut and Marion Videau, *Symmetric Boolean functions*, IEEE Trans. Information Theory **51** (2005), no. 8, 2791–2811.

[CVJ] Titouan Coladon, Vanessa Vitse, and Antoine Joux, *OpenF4 implementation*, `https://github.com/nauotit/openf4`.

[Dav87] James H. Davenport, *Looking at a Set of Equations*, Tech. report, School of Mathematical Sciences, University of Bath, 1987.

[dB12] Marzio de Biasi, *Binary Puzzle is NP-Complete*, July 2012.

[DCG+01] Ed Dawson, Andrew Clark, Helen Gustafson, Bill Millan, and Leonie Sipmson, *Evaluation of TOYOCRYPT-HS1*, Tech. report, Information Security Research Centre, Queensland University of Technology, 2001.

[DCP+17] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang, *Gui - Algorithm Specification and Documentation*, Tech. report, National Institute for Standards and Technology (NIST), 2017.

[DGG+23] Jintai Ding, Boru Gong, Hao Guo, Xiaoou He, Yi Jin, Yuansheng Pan, Dieter Schmidt, Chengdong Tao, Danli Xie, Bo-Yin Yang, and Ziyu Zhao, *TUOV: Triangular Unbalanced Oil and Vinegar*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[DGPS18] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann, SINGULAR *4-1-1 — A computer algebra system for polynomial computations*, `http://www.singular.uni-kl.de`, 2018.

[DH76] Whitfield Diffie and Martin E. Hellman, *New directions in cryptography*, IEEE Trans. Information Theory **22** (1976), no. 6, 644–654.

[Din04] Jintai Ding, *A New Variant of the Matsumoto-Imai Cryptosystem through Perturbation*, in Bao et al. [BDZ04], pp. 305–318.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé, *CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018** (2018), no. 1, 238–268.

[DPW14] Jintai Ding, Albrecht Petzoldt, and Lih-chung Wang, *The Cubic Simple Matrix Encryption Scheme*, in Mosca [Mos14], pp. 76–87.

[DS05] Jintai Ding and Dieter Schmidt, *Rainbow, a New Multivariable Polynomial Signature Scheme*, Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings (John Ioannidis, Angelos D. Keromytis, and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 3531, 2005, pp. 164–175.

[dt16] The FPLLL development team, *fplll, a lattice reduction library*, Available at `https://github.com/fplll/fplll`, 2016.

[DY07] Jintai Ding and Bo-Yin Yang, *Multivariates Polynomials for Hashing*, Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5, 2007, Revised Selected Papers (Dingyi Pei, Moti Yung, Dongdai Lin, and Chuankun Wu, eds.), Lecture Notes in Computer Science, vol. 4990, Springer, 2007, pp. 358–371.

[ECC97] *The Certicom ECC Challenge*, `https://www.certicom.com/index.php/the-certicom-ecc-challenge`, 1997.

[EF17] Christian Eder and Jean-Charles Faugère, *A survey on signature-based algorithms for computing Gröbner bases*, J. Symb. Comput. **80** (2017), 719–784.

[EP10] Christian Eder and John Edward Perry, *F5C: A variant of Faugère's F5 algorithm with reduced Gröbner bases*, J. Symb. Comput. **45** (2010), no. 12, 1442–1458.

[EP11] Christian Eder and John Edward Perry, *Signature-based algorithms to compute Gröbner bases*, Symbolic and Algebraic Computation, International Symposium, ISSAC 2011 (co-located with FCRC 2011), San Jose, CA, USA, June 7-11, 2011, Proceedings (Éric Schost and Ioannis Z. Emiris, eds.), ACM, 2011, pp. 99–106.

[ES03] Niklas Eén and Niklas Sörensson, *An Extensible SAT-solver*, Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers (Enrico Giunchiglia and Armando Tacchella, eds.), Lecture Notes in Computer Science, vol. 2919, Springer, 2003, pp. 502–518.

[Eve87] Jan-Hendrik Evertse, *Linear Structures in Blockciphers*, Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings (David Chaum and Wyn L. Price, eds.), Lecture Notes in Computer Science, vol. 304, Springer, 1987, pp. 249–266.

[Fau99] Jean-Charles Faugère, *A new efficient algorithm for computing Gröbner bases (F4).*, Journal of Pure and Applied Algebra **139** (1999), no. 1–3, 61–88.

[Fau02] Jean Charles Faugère, *A New Efficient Algorithm for Computing GrÖBner Bases Without Reduction to Zero (F5)*, Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (New York, NY, USA), ISSAC '02, ACM, 2002, pp. 75–83.

[Fau10] Jean-Charles Faugère, *FGb: A Library for Computing Gröbner Bases*, Mathematical Software - ICMS 2010, Third International Congress on Mathematical Software, Kobe, Japan, September 13-17, 2010. Proceedings (Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, eds.), Lecture Notes in Computer Science, vol. 6327, Springer, 2010, pp. 84–87.

[FB09] Giordano Fusco and Eric Bach, *Phase transition of multivariate polynomial systems*, Mathematical Structures in Computer Science **19** (2009), 9–23.

[Feo17] Luca De Feo, *Mathematics of Isogeny Based Cryptography*, CoRR **abs/1711.04062** (2017).

[FGLM93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora, *Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering*, J. Symb. Comput. **16** (1993), no. 4, 329–344.

[FGP+15] Jean-Charles Faugère, Danilo Gligoroski, Ludovic Perret, Simona Samardjiska, and Enrico Thomae, *A Polynomial-Time Key-Recovery Attack on MQQ Cryptosystems*, Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings (Jonathan Katz, ed.), Lecture Notes in Computer Science, vol. 9020, Springer, 2015, pp. 150–174.

[FIH+23] Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Tsuyoshi Takagi, Kan Yasuda, Toshiyuki Miyazawa, Tsunekazu Saito, and Akira Nagai, *QR-UOV Specification*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[FJ03] Jean-Charles Faugère and Antoine Joux, *Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases*, in Boneh [Bon03], pp. 44–60.

[FLP08] Jean-Charles Faugère, Françoise Levy-dit-Vehel, and Ludovic Perret, *Cryptanalysis of MinRank*, Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings (David A. Wagner, ed.), Lecture Notes in Computer Science, vol. 5157, Springer, 2008, pp. 280–296.

[FOPT10] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich, *Algebraic Cryptanalysis of McEliece Variants with Compact Keys*, Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings (Henri Gilbert, ed.), Lecture Notes in Computer Science, vol. 6110, Springer, 2010, pp. 279–298.

[FPR17a] Jean-Charles Faugère, Ludovic Perret, and Jocelyn Ryckeghem, *DualModeMS : A Dual Mode for Multivariate-based Signature*, Tech. report, National Institute for Standards and Technology (NIST), 2017.

[FPR+17b] Jean-Charles Faugère, Ludovic Perret, Jocelyn Ryckeghem, A. Casanova, Jacques Patarin, and Gilles Macario-Rat, *GeMSS : A Great Multivariate Short Signature*, Tech. report, National Institute for Standards and Technology (NIST), 2017.

[FR23] Thibauld Feneuil and Matthieu Rivain, *MQOM: MQ on my Mind*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[Gam84] Taher El Gamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings (G. R. Blakley and David Chaum, eds.), Lecture Notes in Computer Science, vol. 196, Springer, 1984, pp. 10–18.

[GCF+23] Louis Goubin, Benoît Cogliati, Jean-Charles Faugère, Pierre-Alain Fouque, Robin Larrieu, Gilles Macario-Rat, Brice Minaud, and Jacques Patarin, *PROV: PRovable unbalanced Oil and Vinegar*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[GEG+17] Ambros Gleixner, Leon Eifler, Tristan Gally, Gerald Gamrath, Patrick Gemander, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Stefan Vigerske, Dieter Weninger, Jonas T. Witt, and Jakob Witzig, *The SCIP Optimization Suite 5.0*, Tech. Report 17-61, ZIB, Takustr. 7, 14195 Berlin, 2017.

[GGV10] Shuhong Gao, Yinhua Guan, and Frank Volny, IV, *A New Incremental Algorithm for Computing Groebner Bases*, Proceedings of

the 2010 International Symposium on Symbolic and Algebraic Computation (New York, NY, USA), ISSAC '10, ACM, 2010, pp. 13–19.

[GIW16] Shuhong Gao, Frank Volny IV, and Mingsheng Wang, *A new framework for computing Gröbner bases*, Math. Comput. **85** (2016), no. 297, 449–465.

[GM88] Rüdiger Gebauer and H. Michael Möller, *On an Installation of Buchberger's Algorithm*, J. Symb. Comput. **6** (1988), no. 2/3, 275–286.

[Gol94] Jovan Dj. Golic, *Linear Cryptanalysis of Stream Ciphers*, in Preneel [Pre95], pp. 154–169.

[Gur16] Gurobi Optimization, Inc., *Gurobi Optimizer Reference Manual*, 2016.

[HA10] Amir Hashemi and Gwénolé Ars, *Extended F_5 criteria*, J. Symb. Comput. **45** (2010), no. 12, 1330–1340.

[HB13] Heliang Huang and Wansu Bao, *Middle-Solving F4 to Compute Grobner bases for Cryptanalysis over GF(2)*, CoRR **abs/1310.2332** (2013).

[HCN19] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg, *Multidimensional Linear Cryptanalysis*, J. Cryptology **32** (2019), no. 1, 1–34.

[HDRM23] Solane El Hirch, Joan Daemen, Raghvendra Rohit, and Rusydi H. Makarim, *Twin Column Parity Mixers and Gaston - A New Mixing Layer and Permutation*, Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III (Helena Handschuh and Anna Lysyanskaya, eds.), Lecture Notes in Computer Science, vol. 14083, Springer, 2023, pp. 475–506.

[Hel94] Tor Helleseth (ed.), *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, Lecture Notes in Computer Science, vol. 765, Springer, 1994.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman, *NTRU: A Ring-Based Public Key Cryptosystem*, Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings (Joe Buhler, ed.), Lecture Notes in Computer Science, vol. 1423, Springer, 1998, pp. 267–288.

[IM85] Hideki Imai and Tsutomu Matsumoto, *Algebraic Methods for Constructing Asymmetric Cryptosystems*, Algebraic Algorithms and Error-Correcting Codes, 3rd International Conference, AAECC-3, Grenoble, France, July 15-19, 1985, Proceedings (Jacques Calmet, ed.), Lecture Notes in Computer Science, vol. 229, Springer, 1985, pp. 108–119.

[JDF11] David Jao and Luca De Feo, *Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies*, Post-Quantum Cryptography (PQCrypto 2011), Springer, 2011, pp. 19–34.

[JJGvD13] Christopher Jefferson, Peter Jeavons, Martin James Green, and Marc R. C. van Dongen, *Representing and solving finite-domain constraint problems using systems of polynomials*, Ann. Math. Artif. Intell. **67** (2013), no. 3-4, 359–382.

[JV11] Antoine Joux and Vanessa Vitse, *A Variant of the F4 Algorithm*, Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings (Aggelos Kiayias, ed.), Lecture Notes in Computer Science, vol. 6558, Springer, 2011, pp. 356–375.

[JV17] Antoine Joux and Vanessa Vitse, *A Crossbred Algorithm for Solving Boolean Polynomial Systems*, Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017, Warsaw, Poland, September 11-13, 2017, Revised Selected Papers (Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykala, eds.), Lecture Notes in Computer Science, vol. 10737, Springer, 2017, pp. 3–21.

[Ker83] Auguste Kerckhoffs, *La Cryptographie Militaire*, Journal des Sciences Militaires **IX** (1883), 5–83.

[Knu94] Lars R. Knudsen, *Truncated and Higher Order Differentials*, in Preneel [Pre95], pp. 196–211.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin, *Unbalanced Oil and Vinegar Signature Schemes*, in Stern [Ste99], pp. 206–222.

[KR96] Lars R. Knudsen and Matthew J. B. Robshaw, *Non-Linear Approximations in Linear Cryptanalysis*, in Maurer [Mau96], pp. 224–236.

[KR00] Martin Kreuzer and Lorenzo Robbiano, *Computational Commutative Algebra 1*, Springer Berlin Heidelberg, 2000.

[Kra93] David Kravitz, *Digital Signature Algorithm*, U.S. Patent #5,231,668, 1993.

[KS99] Aviad Kipnis and Adi Shamir, *Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization*, Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings (Michael J. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, Springer, 1999, pp. 19–30.

[Lai94] Xuejia Lai, *Additive and Linear Structures of Cryptographic Functions*, in Preneel [Pre95], pp. 75–85.

[Lev73] L. A. Levin, *Universal enumeration problems*, Problemy Peredači Informacii **9** (1973), no. 3, 115–116. MR 0340042

[LH94] Susan K. Langford and Martin E. Hellman, *Differential-Linear Cryptanalysis*, Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings (Yvo Desmedt, ed.), Lecture Notes in Computer Science, vol. 839, Springer, 1994, pp. 17–25.

[Ln23] Ignacio Luengo and Martín Avenda no, *DME: MULTIVARIATE SIGNATURE PUBLIC KEY SCHEME*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[LPT+17] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu, *Beating Brute Force for Systems of Polynomial Equations over Finite Fields*, Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19 (Philip N. Klein, ed.), SIAM, 2017, pp. 2190–2202.

[Luc78] Edouard Lucas, *Théorie des Fonctions Numériques Simplement Périodiques*, American Journal of Mathematics **1** (1878), no. 2, 184–196.

[Lue17] Ignacio Luengo, *DME : A Public Key, Signature, and KEM System Based on Double Exponentiation with Matrix Exponents*, Tech. report, ICMAT and Department of Algebra, Geometry, and Topology, Complutense University of Madrid, Plaza de Las Ciencias S/N, CIUDAD Universitaria 28040 Madrid, Spain, 2017.

[MAAT02] Mohammed Mrayati, Yahya Meer Alam, and M. Hassan At-Tayyan, *Al-Kindi's Treatise on Cryptanalysis*, Series on Arabic Origins of Cryptology, vol. one, KFCRIS & KACST, 2002.

[MAB+21] Carlos Aguilar Melchor, Nicolas Aragon, Emanuele Bellini, Florian Caullery, Rusydi H. Makarim, and Chiara Marcolla, *Constant Time Algorithms for ROLLO-I-128*, SN Comput. Sci. **2** (2021), no. 5, 382.

[Mak18] Andrew O. Makhorin, *GNU Linear Programming Kit*, 2018.

[Map18] Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario, *Maple 2018*, `https://www.maplesoft.com/`, 2018.

[Mat93] Mitsuru Matsui, *Linear Cryptanalysis Method for DES Cipher*, in Helleseth [Hel94], pp. 386–397.

[Mat97] Mitsuru Matsui, *New Block Encryption Algorithm MISTY*, Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings (Eli Biham, ed.), Lecture Notes in Computer Science, vol. 1267, Springer, 1997, pp. 54–68.

[Mau96] Ueli M. Maurer (ed.), *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, Lecture Notes in Computer Science, vol. 1070, Springer, 1996.

[MCD+09] Mohamed Saied Emam Mohamed, Daniel Cabarcas, Jintai Ding, Johannes A. Buchmann, and Stanislav Bulygin, *MXL_3: An Efficient Algorithm for Computing Gröbner Bases of Zero-Dimensional Ideals*, Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers (Dong Hoon Lee and Seokhie Hong, eds.), Lecture Notes in Computer Science, vol. 5984, Springer, 2009, pp. 87–100.

[MDBW09] Mohamed Saied Emam Mohamed, Jintai Ding, Johannes A. Buchmann, and Fabian Werner, *Algebraic Attack on the MQQ Public Key Cryptosystem*, Cryptology and Network Security, 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings (Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, eds.), Lecture Notes in Computer Science, vol. 5888, Springer, 2009, pp. 392–401.

[Mer78] Ralph C. Merkle, *Secure Communications Over Insecure Channels*, Commun. ACM **21** (1978), no. 4, 294–299.

[MH78] Ralph C. Merkle and Martin E. Hellman, *Hiding information and signatures in trapdoor knapsacks*, IEEE Trans. Information Theory **24** (1978), no. 5, 525–530.

[MHT13] Hiroyuki Miura, Yasufumi Hashimoto, and Tsuyoshi Takagi, *Extended Algorithm for Solving Underdefined Multivariate Quadratic Equations*, Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings (Philippe Gaborit, ed.), Lecture Notes in Computer Science, vol. 7932, Springer, 2013, pp. 118–135.

[Min95] S. Minato, *Implicit manipulation of polynomials using zero-suppressed BDDs*, Proceedings the European Design and Test Conference. ED TC 1995, Mar 1995, pp. 449–454.

[MMDB08] Mohamed Saied Emam Mohamed, Wael Said Abd Elmageed Mohamed, Jintai Ding, and Johannes A. Buchmann, *MXL2: Solving Polynomial Equations over GF(2) Using an Improved Mutant Strategy*, Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings (Johannes A. Buchmann and Jintai Ding, eds.), Lecture Notes in Computer Science, vol. 5299, Springer, 2008, pp. 203–215.

[MMR91] Harold F. Mattson, Teo Mora, and T. R. N. Rao (eds.), *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 9th International Symposium, AAECC-9, New Orleans, LA, USA, October 7-11, 1991, Proceedings*, Lecture Notes in Computer Science, vol. 539, Springer, 1991.

[Mon16] David Monniaux, *A Survey of Satisfiability Modulo Theory*, CoRR **abs/1606.04786** (2016).

[Mos14] Michele Mosca (ed.), *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October*

1-3, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8772, Springer, 2014.

[MP08] Sean Murphy and Maura B. Paterson, *A geometric view of cryptographic equation solving*, J. Mathematical Cryptology **2** (2008), no. 1, 63–107.

[MP15] Michael Monagan and Roman Pearce, *A Compact Parallel Implementation of F4*, Proceedings of the 2015 International Workshop on Parallel Symbolic Computation (New York, NY, USA), PASCO '15, ACM, 2015, pp. 95–100.

[MR22] Rusydi H. Makarim and Raghvendra Rohit, *Towards Tight Differential Bounds of Ascon A Hybrid Usage of SMT and MILP*, IACR Trans. Symmetric Cryptol. **2022** (2022), no. 3, 303–340.

[MS89a] Willi Meier and Othmar Staffelbach, *Fast Correlation Attacks on Certain Stream Ciphers*, J. Cryptology **1** (1989), no. 3, 159–176.

[MS89b] Willi Meier and Othmar Staffelbach, *Nonlinearity Criteria for Cryptographic Functions*, Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings (Jean-Jacques Quisquater and Joos Vandewalle, eds.), Lecture Notes in Computer Science, vol. 434, Springer, 1989, pp. 549–562.

[MS17] Rusydi H. Makarim and Marc Stevens, *M4GB: An Efficient Gröbner-Basis Algorithm*, Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017, Kaiserslautern, Germany, July 25-28, 2017 (Michael A. Burr, Chee K. Yap, and Mohab Safey El Din, eds.), ACM, 2017, pp. 293–300.

[MT14] Rusydi H. Makarim and Cihangir Tezcan, *Relating Undisturbed Bits to Other Properties of Substitution Boxes*, Lightweight Cryptography for Security and Privacy - Third International Workshop, LightSec 2014, Istanbul, Turkey, September 1-2, 2014, Revised Selected Papers (Thomas Eisenbarth and Erdinç Öztürk, eds.), Lecture Notes in Computer Science, vol. 8898, Springer, 2014, pp. 109–125.

[MvOV96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[Nat77] National Bureau of Standards (NBS), *Data Encryption Standard*, U.S. Department of Commerce, Washington DC, publication 46 ed., 1977.

[Neu13] Severin Neumann, *A modified parallel F4 algorithm for shared and distributed memory architectures*, 5th International Symposium on Symbolic Computation in Software Science, SCSS 2013, Castle of Hagenberg, Austria (Laura Kovács and Temur Kutsia, eds.), EPiC Series in Computing, vol. 15, EasyChair, 2013, pp. 70–80.

[Nin17] Kai-Chun Ning, *An Adaptation of the Crossbred Algorithm for Solving Multivariate Quadratic Systems over* $\mathbb{F}_2$ *on GPUs*, November 2017, Available at `https://pure.tue.nl/ws/portalfiles/portal/91105984/NING.K_parallel_cb_v103.pdf`.

[NIS01] NIST, *FIPS 197 - Advanced Encryption Standard (AES)*, November 2001.

[NIS20] NIST, *Post-Quantum Cryptography, Round 3 Submissions*, July 2020.

[NIS22] NIST, *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*, October 2022.

[NNY18] Ruben Niederhagen, Kai-Chun Ning, and Bo-Yin Yang, *Implementing Joux-Vitse's Crossbred Algorithm for Solving $${\mathcal M\mathcal Q}$$ Systems over* $\mathbf{F}_2$ *on GPUs*, Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings (Tanja Lange and Rainer Steinwandt, eds.), Lecture Notes in Computer Science, vol. 10786, Springer, 2018, pp. 121–141.

[Nyb92] Kaisa Nyberg, *On the Construction of Highly Nonlinear Permutations*, Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings (Rainer A. Rueppel, ed.), Lecture Notes in Computer Science, vol. 658, Springer, 1992, pp. 92–98.

[Nyb93] Kaisa Nyberg, *Differentially Uniform Mappings for Cryptography*, in Helleseth [Hel94], pp. 55–64.

[Pat96] Jacques Patarin, *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms*, in Maurer [Mau96], pp. 33–48.

[PBD14] Jaiberth Porras, John Baena, and Jintai Ding, *ZHFE, a New Multivariate Public Key Encryption Scheme*, in Mosca [Mos14], pp. 229–245.

[PCDY17] Albrecht Petzoldt, Ming-Shing Chen, Jintai Ding, and Bo-Yin Yang, *HMFEv - An Efficient Multivariate Signature Scheme*, Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings (Tanja Lange and Tsuyoshi Takagi, eds.), Lecture Notes in Computer Science, vol. 10346, Springer, 2017, pp. 205–223.

[PCF+23] Jacques Patarin, Benoît Cogliati, Jean-Charles Faugère, Pierre-Alain Fouque, Louis Goubin, Robin Larrieu, Gilles Macario-Rat, and Brice Minaud, *Vox Specification*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[PFH+] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang, *FALCON: Fast-Fourier Lattice-based Compact Signatures over NTRU*.

[Pre95] Bart Preneel (ed.), *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, Lecture Notes in Computer Science, vol. 1008, Springer, 1995.

[Pre11] Bart Preneel, *Modes of Operation of a Block Cipher*, Encyclopedia of Cryptography and Security, 2nd Ed. (Henk C. A. van Tilborg and Sushil Jajodia, eds.), Springer, 2011, pp. 789–794.

[Pro99] Third Generation Partnership Project, *KASUMI Specification*, Tech. Report version 1.0, Security Algorithms Group of Experts (SAGE), 1999.

[Rad06] Håvard Raddum, *Cryptanalytic Results on Trivium*, Tech. report, eSTREAM Report 2006/039, 2006.

[RN10] Stuart Russell and Peter Norvig, *Artifical Intelligence, a Modern Approach*, third ed., Series in Artificial Intelligence, Prentice Hall, 2010.

[Rod23a] Borja Gómez Rodríguez, *3WISE: Cubic Element-Wise trapdoor based MPKC cryptosystem*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[Rod23b] Borja Gómez Rodríguez, *HPPC: Hidden Product of Polynomial Composition*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[RS06] Håvard Raddum and Igor A. Semaev, *New Technique for Solving Sparse Equation Systems*, IACR Cryptology ePrint Archive **2006** (2006), 475.

[RS08] Håvard Raddum and Igor A. Semaev, *Solving Multiple Right Hand Sides linear equations*, Des. Codes Cryptography **49** (2008), no. 1-3, 147–160.

[RS12] Bjarke Hammersholt Roune and Michael Stillman, *Practical Gröbner basis computation*, in van der Hoeven and van Hoeij [vdHvH12], pp. 203–210.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Commun. ACM **21** (1978), no. 2, 120–126.

[RSA91] *The RSA Factoring Challenge*, `http://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-factoring-challenge.htm`, 1991.

[Sak98] Shojiro Sakata, *Gröbner Bases and Coding Theory*, London Mathematical Society Lecture Note Series, p. 205–220, Cambridge University Press, 1998.

[SDGM00] Leonie Ruth Simpson, Ed Dawson, Jovan Dj. Golic, and William Millan, *LILI Keystream Generator*, in Stinson and Tavares [ST01], pp. 248–261.

[SG14] Simona Samardjiska and Danilo Gligoroski, *Linearity Measures for Multivariate Cryptography*, SECURWARE 2014, The Eighth International Conference on Emerging Security Information, Systems and Technologies, International Academy, Research and Industry Association (IARIA), 2014, pp. 157–166.

[Sha49] Claude E. Shannon, *Communication theory of secrecy systems*, Bell Labs Technical Journal **28** (1949), no. 4, 656–715.

[Sho97] Peter W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM J. Comput. **26** (1997), no. 5, 1484–1509.

[Sho08] Victor Shoup, *A Computational Introduction to Number Theory and Algebra*, vol. 2, Cambridge University Press, 2008.

[Sie85] Thomas Siegenthaler, *Decrypting a Class of Stream Ciphers Using Ciphertext Only*, IEEE Trans. Computers **34** (1985), no. 1, 81–85.

[Sin00] Simon Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Anchor Books, 2000.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia, *Extending SAT Solvers to Cryptographic Problems*, Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings (Oliver Kullmann, ed.), Lecture Notes in Computer Science, vol. 5584, Springer, 2009, pp. 244–257.

[Soo16] Mate Soos, *The CryptoMiniSat 5 set of solvers at SAT Competition 2016*, Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions - SAT Competition 2016, 2016, Available at `https://github.com/msoos/cryptominisat`, p. 28.

[SPK17] Kyung-Ah Shim, Cheol-Min Park, and Aeyoung Kim, *HiMQ-3 : A High Speed Signature Scheme based on Multivariate Quadratic Equations*, Tech. report, National Institute for Standards and Technology (NIST), 2017.

[SSM+09] Massimiliano Sala, Shojiro Sakata, Teo Mora, Carlo Traverso, and Ludovic Perret (eds.), *Gröbner Bases, Coding, and Cryptography*, Springer Berlin Heidelberg, 2009.

[ST01] Douglas R. Stinson and Stafford E. Tavares (eds.), *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings*, Lecture Notes in Computer Science, vol. 2012, Springer, 2001.

[ST23] Kosuke Sakata and Tsuyoshi Takagi, *An Efficient Algorithm for Solving the MQ Problem using Hilbert Series*, Cryptology ePrint Archive, Paper 2023/1650, 2023, `https://eprint.iacr.org/2023/1650`.

[Ste99] Jacques Stern (ed.), *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, Lecture Notes in Computer Science, vol. 1592, Springer, 1999.

[Ste24] Matthias Johann Steiner, *The Complexity of Algebraic Algorithms for LWE*, Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part III (Marc Joye and Gregor Leander, eds.), Lecture Notes in Computer Science, vol. 14653, Springer, 2024, pp. 375–403.

[SVP10] *The Lattice Challenge*, `http://www.latticechallenge.org/`, 2010.

[TDTD13] Chengdong Tao, Adama Diene, Shaohua Tang, and Jintai Ding, *Simple Matrix Scheme for Encryption*, Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings (Philippe Gaborit, ed.), Lecture Notes in Computer Science, vol. 7932, Springer, 2013, pp. 231–242.

[The18] The Sage Developers, *SageMath, the Sage Mathematics Software System (Version 8.3.0)*, 2018, `http://www.sagemath.org`.

[UM17] Putranto H. Utomo and Rusydi H. Makarim, *Solving a Binary Puzzle*, Mathematics in Computer Science **11** (2017), no. 3-4, 515–526.

[UP15] Putranto H. Utomo and Ruud Pellikaan, *Binary Puzzles as an Erasure Decoding Problem*, Proceedings of the 36th Symposium on Information Theory in the Benelux and the 5th Joint WIC/IEEE Symposium on Information Theory and Signal Processing in the Benelux (Jérémie Roland and François Horlin, eds.), 2015.

[UP17] Putranto H. Utomo and Ruud Pellikaan, *Binary Puzzles as a SAT Problem*, Proceedings of the 2017 Symposium on Information Theory and Signal Processing in the Benelux (Richard Heusdens and Jos Weber, eds.), 2017.

[Uto17] Putranto Utomo, *Satisfiability modulo theory and binary puzzle*, Journal of Physics: Conference Series **855** (2017), no. 1, 012057.

[vdHvH12] Joris van der Hoeven and Mark van Hoeij (eds.), *International Symposium on Symbolic and Algebraic Computation, ISSAC'12, Grenoble, France - July 22 - 25, 2012*, ACM, 2012.

[VDI24] Damien Vidal, Claire Delaplace, and Sorina Ionica, *An analysis of the Crossbred Algorithm for the MQ Problem*, IACR Commun. Cryptol. **1** (2024), no. 3, 36.

[vzG15] Joachim von zur Gathen, *CryptoSchool*, Springer, 2015.

[Wan98] Dongming Wang, *Gröbner Bases Applied to Geometric Theorem Proving and Discovering*, London Mathematical Society Lecture Note Series, p. 281–302, Cambridge University Press, 1998.

[WCD+23] Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang, *SNOVA Proposal for NISTPQC: Digital Signature Schemes project*, Tech. report, National Institute for Standards and Technology (NIST), 2023.

[WHL+05] Lih-Chung Wang, Yuh-Hua Hu, Feipei Lai, Chun-yen Chou, and Bo-Yin Yang, *Tractable Rational Map Signature*, Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings (Serge Vaudenay, ed.), Lecture Notes in Computer Science, vol. 3386, Springer, 2005, pp. 244–257.

[Wie86] Douglas H. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inf. Theory **32** (1986), no. 1, 54–62.

[Wol05] Christopher Wolf, *Multivariate Quadratic Polynomials in Public-Key Cryptography*, Ph.D. thesis, Katholieke Universiteit Leuven, November 2005.

[YCC04] Bo-Yin Yang, Jiun-Ming Chen, and Nicolas Courtois, *On Asymptotic Security Estimates in XL and Gröbner Bases-Related Algebraic Cryptanalysis*, Information and Communications Security, 6th International Conference, ICICS 2004, Malaga, Spain, October 27-29, 2004, Proceedings (Javier López, Sihan Qing, and Eiji Okamoto, eds.), Lecture Notes in Computer Science, vol. 3269, Springer, 2004, pp. 401–413.

[YDH+15a] Takanori Yasuda, Xavier Dahan, Yun-Ju Huang, Tsuyoshi Takagi, and Kouichi Sakurai, *MQ Challenge: Hardness Evaluation of Solving Multivariate Quadratic Problems*, IACR Cryptology ePrint Archive **2015** (2015), 275.

[YDH+15b] Takanori Yasuda, Xavier Dahan, Yun-Ju Huang, Tsuyoshi Takagi, and Kouichi Sakurai, *A multivariate quadratic challenge toward post-quantum generation cryptography*, ACM Comm. Computer Algebra **49** (2015), no. 3, 105–107.

Notations

$\mathbb{Z}$	Ring of integers	49
$\mathbb{Z}_{\geq 0}$	The set of non-negative integers	49
$\mathbb{F}$	A field	49
$\mathbb{F}_q$	Finite field of order q	49
$\deg(f)$	The degree of the polynomial f	51
$\mathbb{F}[x_1, \ldots, x_n]$	An n-variable polynomial ring over the field $\mathbb{F}$	50
$\mathcal{R}$	The ring $\mathbb{F}[x_1, \ldots, x_n]$ unless explicitly stated otherwise	50
$\langle f_1, \ldots, f_m \rangle$	Ideal generated by $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$	51
$x^\alpha \mid x^\beta$	Monomial x^α divides x^β	49
$\mathsf{M}(f)$	The set of monomials of $f \in \mathbb{F}[x_1, \ldots, x_n]$	50
$\mathsf{T}(f)$	The set of terms of $f \in \mathbb{F}[x_1, \ldots, x_n]$	50
$\mathsf{M}(F)$	The set of monomials of $F \subseteq \mathbb{F}[x_1, \ldots, x_n]$	50
$\mathsf{T}(F)$	The set of terms of $F \subseteq \mathbb{F}[x_1, \ldots, x_n]$	50
$\mathsf{LC}(f)$	The leading coefficient of a polynomial $f \neq 0$	53
$\mathsf{LM}(f)$	The leading monomial of a polynomial $f \neq 0$	53
$\mathsf{LT}(f)$	The leading term of a polynomial $f \neq 0$	53
$\mathsf{Tail}(f)$	The tail of a polynomial $f \neq 0$	53
$\overline{f}^G$	The remainder after division of f by an ordered set G	55
$\mathbf{V}(f_1, \ldots, f_m)$	The affine variety defined by the polynomials $f_1, \ldots, f_m$	59
$\mathbf{I}(V)$	The ideal of affine variety V	60
I_ℓ	The ℓ-th elimination ideal	60
$\mathsf{Spoly}(f, g)$	S-polynomial of $f \neq 0$ and $g \neq 0$	65
$\mathsf{LCM}(a, b)$	The least common multiple of a and b	65
$f \underset{G}{\longrightarrow} r$	f reduces to r modulo G	67
$\widetilde{F}$	(reduced) row echelon form of $F \subseteq \mathcal{R}$	72
$\mathrm{wt}(u)$	The Hamming weight of $u \in \mathbb{F}_2^n$	120
$v \preceq w$	The vector v is covered by w	120
φ_f	The pseudo Boolean function of f	121
$v(f)$	The simplified value vector of a symmetric Boolean function f	121
$\sigma_{i,n}$	The i-th elementary symmetric polynomial in n variables	122
$\mathsf{FP}(\mathcal{R})$	The set of field polynomials of $\mathcal{R} = \mathbb{F}_q[x_1, \ldots, x_n]$	133
$\boldsymbol{v} \parallel \boldsymbol{w}$	Concatenation of vector $\boldsymbol{v} \in \mathbb{F}_2^n$ and $\boldsymbol{w} \in \mathbb{F}_2^m$	133
$\boldsymbol{v} \cdot \boldsymbol{w}$	The dot product of $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{F}_2^n$	133
$S_{\boldsymbol{b}}$	The function $\boldsymbol{b} \cdot S(x)$ where $S : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$	133
$\mathcal{F}_f, \mathcal{W}_f$	The Fourier and Walsh transform of $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ respectively	134
$\lambda_S(\boldsymbol{a}, \boldsymbol{b})$	The linear approximation table of S	135
$\lambda I_S(\boldsymbol{a}, \boldsymbol{b})$	The linear ideal of S	135
$\delta_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$	The difference distribution table of S	138
$\delta I_S(\boldsymbol{\alpha}, \boldsymbol{\beta})$	The differential ideal of S	139

$\tau_S(\boldsymbol{\alpha}, \boldsymbol{b})$	The autocorrelation table of S	141
$\tau I_S(\boldsymbol{\alpha}, \boldsymbol{b})$	The autocorrelation ideal of S	142

Index

List of Algorithms

List of Figures

List of Tables

English Summary

Let $\mathbb{F}[x_1, \ldots, x_n]$ be a multivariate polynomial ring in variables $x_1, \ldots, x_n$ with coefficients in a field $\mathbb{F}$ and a chosen monomial ordering. Let $I \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be an ideal such that $I \neq \{0\}$. A finite subset G of the ideal I is said to be a Gröbner basis of I if for all $f \in I \setminus \{0\}$ there exists a polynomial $g \in G$ such that the leading term of g divides the leading term of f.

The primary subject in this thesis are Gröbner-basis algorithms, i.e., algorithms that compute a Gröbner basis for a given ideal $I \neq \{0\}$ of $\mathbb{F}[x_1, \ldots, x_n]$. In addition to the interest in the algorithms themselves, the work in this thesis also covers applications of Gröbner-basis algorithms. For instance, to solve systems of equations of $\mathbb{F}[x_1, \ldots, x_n]$ over a finite field $\mathbb{F}$, which often arise in the security analysis of cryptographic schemes.

In Chapter 5 we introduce a variant Gröbner basis algorithm called M4GB. The M4GB algorithm is described in a way that unifies high-level description together with its main implementation aspects, with the goal to present it without obscuring the nature of its implementation. It is namely designed to consistently operate on and maintain polynomials in tail-reduced form with respect to the current intermediate basis. Our new algorithm relies on a new recursive reduction algorithm for polynomials given in the form $\mathtt{t} \cdot f$, where $\mathtt{t}$ and f are a term and a non-zero polynomial in $\mathbb{F}[x_1, \ldots, x_n]$ respectively.

In Chapter 6 we present the application of M4GB to solve several concrete challenges of multivariate quadratic (MQ) problems over finite fields. In particular M4GB sets a new record in the Fukuoka MQ challenge* for parameters that represent the public-key in MQ-based digital signature algorithms. In the same chapter we also analyze the expected required CPU time and memory to solver larger parameters.

In Chapter 7 we demonstrate another application of Gröbner-basis algorithms to solve binary puzzles. A binary puzzle is a Sudoku-like puzzle with an entry in each cell taken from the set $\{0, 1\}$ instead of $\{0, 1, \ldots, 9\}$. A binary puzzle is solvable if there exists an assignment for each cell that satisfies all three constraints:

1. there exists no three (3) consecutive cells, both vertically and horizontally, which are filed with the same value;
2. the number of zeros and ones in each row and column must be equal;
3. every two distinct rows (resp. columns) must not be equal.

The main idea to solve binary puzzles using Gröbner bases algorithms is to view it as a constraint satisfaction problem where each constraint above is represented by a set of multivariate polynomials. We introduce two approaches for the derivation of such a set of polynomials that represent the solvability of a binary puzzle. The first approach is to model the constraints as polynomials over $\mathbb{F}_2$. The second one is to model the polynomials over $\mathbb{Z}$ or $\mathbb{Q}$. The advantage of the latter approach than the former is it reduces the degree of some polynomials at the expense of defining the coefficients over different rings. A solution for the binary puzzle is obtained by computing a Gröbner basis of the ideal generated by the polynomials that represent the three constraints.

* `https://www.mqchallenge.org`

In Chapter 8 we establish connections among cryptographic properties of vectorial Boolean functions and computational commutative algebra. We propose ideal-theoretic characterization for notions related to the (non)linearity, differential, and autocorrelation of vectorial Boolean functions. The three mentioned notions are related to the linear, differential, and differential-linear cryptanalysis of symmetric-key cryptography. The primary contributions are the construction of ideals that are useful to analyse the algebraic implications of linear, differential, and differential-linear cryptanalysis on a vectorial Boolean function. We also exhibit new approaches to compute the bias of a linear approximation (including Walsh and Fourier transform), the probability of a differential, and the autocorrelation of the component functions using Gröbner bases algorithms.

Nederlandse Samenvatting

Neem een polynomiale ring $\mathbb{F}[x_1, \ldots, x_n]$ in variabelen $x_1, \ldots, x_n$ met coëfficienten in een lichaam $\mathbb{F}$ en een gekozen monomiale ordening. Neem een ideaal $I \subseteq \mathbb{F}[x_1, \ldots, x_n]$ met $I \neq \{0\}$. Een eindige deelverzameling G van het ideaal I wordt een Gröbner basis van I genoemd indien voor alle $f \in I \setminus \{0\}$ er een polynoom $g \in G$ bestaat zodat de leidende term van g de leidende term van f deelt.

Het hoofdonderwerp van dit proefschrift zijn Gröbner-basis algoritmen, dat wil zeggen algoritmen die een Gröbner basis berekenen voor een gegeven ideaal $I \neq \{0\}$ van $\mathbb{F}[x_1, \ldots, x_n]$. Naast de interesse in de algoritmen zelf, worden ook toepassingen van Gröbner-basis algoritmen behandeld in dit proefschrift. Bijvoorbeeld om systemen van vergelijkingen van $\mathbb{F}[x_1, \ldots, x_n]$ over een eindig lichaam $\mathbb{F}$ op te lossen, welke vaak voorkomen in de veiligheidsanalyze van cryptografische systemen.

In hoofdstuk 5 introduceren we een variant Gröbner-basis algoritme genaamd M4GB. Het M4GB algoritme is beschreven op een wijze waarbij de hoog niveau beschrijving geintegreerd is met belangrijke implementatie aspecten, met het doel het algoritme te presenteren zonder het karakter van de implementatie te verliezen. Dit algoritme is namelijk ontworpen om altijd te berekenen op, en bijwerken van, polynomen in staart-gereduceerde vorm met respect tot de huidige tussentijdse basis. Ons nieuwe algoritme berust op een nieuw recursief reduceer algoritme voor polynomen gegeven in de vorm $\mathtt{t} \cdot f$, waar $\mathtt{t}$ en f respectievelijk een term en een niet-nul polynoom zijn in $\mathbb{F}[x_1, \ldots, x_n]$.

In hoofdstuk 6 behandelen we de applicatie van M4GB om meerdere concrete uitdagingen van multivariate kwadratische (MQ) vergelijkingen over eindige lichamen. In het bijzonder heeft M4GB een nieuwe record gezet in de Fukuoka MQ Challenge* voor parameters die gerelateerd zijn aan MQ-gebaseerde cryptografische digitale handtekening systemen. In hetzelfde hoofdstuk analyzeren we ook de verwachte benodigde CPU tijd en geheugen om grotere parameters te kunnen oplossen.

In hoofdstuk 7 demonstreren we een andere applicatie van Gröbner-basis algoritmen om binaire puzzels op te lossen. Een binaire puzzel een Sodokuachtige puzzel waar een waarde in elke cel alleen 0 of 1 mag zijn, in plaats van 1 tot en met 9. Een binaire puzzel is oplosbaar als er elke cel met een waarde gevuld kan worden onder de volgende drie condities:

1. er bestaan geen drie (3) opeenvolgende cellen, danwel verticaal of horizontaal, die dezelfde waarde hebben;
2. het aantal nullen en aantal enen in elke rij of kolom moet gelijk zijn;
3. er mogen geen twee rijen, danwel twee kolommen, gelijk aan elkaar zijn.

De kerngedachte om binaire puzzels op te lossen met Gröbner-basis algoritmen is om de puzzel te zien als een Constraint Satisfaction Problem, waar elke bovenstaande conditie wordt beschreven door een set multivariate polynomen. We introduceren twee methoden om een dergelijke set van polynomen die de oplosbaarheid modeleren te bepalen. De eerste methode is om de condities te modeleren als polynomen over het eindige lichaam $\mathbb{F}_2$. De tweede methode

* `https://www.mqchallenge.org`

modeleert polynomen over $\mathbb{Z}$ of $\mathbb{Q}$. Het voordeel van de tweede methode over de eerste methode is dat het de graad van bepaalde polynomen reduceert ten koste van coefficienten over andere ringen. Een oplossing voor de binaire puzzel kan worden uitgerekend door een Gröbner basis uit te rekenen voor het ideaal gegenereerd door de set polynomen die de drie condities modeleren.

In hoofdstuk 8 leggen we verbanden tussen cryptografische eigenschappen van vectoriële Booleaanse functies en computationele commutatieve algebra. We stellen een ideaaltheoretische karakterisering voor voor noties gerelateerd aan de (niet)lineariteit, differentiaal en autocorrelatie van vectoriële Booleaanse functies. De drie genoemde begrippen houden verband met de lineaire, differentiële en differentieel-lineaire cryptoanalyse van cryptografie primitieven met symmetrische sleutels. De belangrijkste bijdragen zijn de constructie van idealen die nuttig zijn om de algebraïsche implicaties van lineaire, differentiële en differentieellineaire cryptanalyse op een vectoriële Booleaanse functie te analyseren. We laten ook nieuwe benaderingen zien om de bias van een lineaire benadering (inclusief Walsh- en Fourier-transformatie), de waarschijnlijkheid van een differentiaal en de autocorrelatie van de componentfuncties te berekenen met behulp van Gröbner-basis algoritmen.

Acknowledgments

I would like to thank my supervisor Ronald Cramer and my co-supervisor Marc Stevens. The opportunity to pursue a PhD degree in the Netherlands has opened up a whole new world in my educational, professional, and personal experience. The thought of doing a PhD in Mathematics was something that never crossed my mind, let alone doing it in the Netherlands. I especially want to thank Marc for multiple discussions and feedbacks that improved my research ability as well as my technical knowledge on the modern C++ features.

I would like to thank the financial support from the Mathematical Institute, University Leiden under the ALGANT (Algebra, Geometry and Number Theory) PhD program. A special thanks to Peter Stevenhagen for coordinating and arranging the funding.

I want to thank my fellow PhD students at the CWI Cryptology Group for the enjoyable discussions over coffee break and table-tennis game. A special thanks to Wessel van Woerden for proofreading part of this thesis and providing useful feedbacks.

I want to thank Putranto Hadi Utomo as one of my collaborator. Our brief meeting in Utrecht turned into a collaborative work after I realized that the problem of solving a binary puzzle can be reduced to the problem of solving a system of polynomial equations. On this part I would also like to thank Ruud Pellikaan for his feedbacks on the simplification of some polynomials that represent the constraints.

I want to express my gratitude also to Ami Muhammad Djuned and family (Ameh Fatma, Hana, Maryam) and Torik Djuned and family (Hafida, Rayyan, Adam, Sofia). Thank you for helping me to settle down in the Netherlands and for always being there whenever I needed help.

My PhD life in the Netherlands would be completely different without the presence of Om Tezar Saputra and family (Teh Intan, Aiden, and Alisha). There is no way I could thank all of you enough for accepting me as part of your family. Thank you for the patience, for the help, for the discussion over dinner, for the food, for the jokes, for the trip together, for the lessons on life and most importantly for the Pro Evolution Soccer game night. The warmth of your family is an antidote for me to fight against homesickness and gloomy cold winter in the Netherlands.

My special thanks also goes to Ahmed Abd-El-Haliem and family, Fuat Özman, Furkan Özman, Abdulaziz Balbaid and family, Said Badjuber, Ibrahim, Adam Belloum, Haryadi and Aya, Damar and Mauril, Mas Fajran and family, the Indonesian students in Leiden (Albert and Dwi) and in Amsterdam (Pram, Hana, Bowo, Widi). I also want to thank Bas Edixhoven for many conversations we had over mathematics, culture, education, and your wish to visit Indonesia again to try to ride a motorcycle. You will be missed greatly.

I want to thank my parents and my brothers Nabiel and Zacky for their support while I was thousands of kilometres away from home.

Finally, for Sarah, Ilyas, and Janan: thank you for your patience and for constantly reminding me that there is a beauty in life outside the realm of rationality and logic.

Curriculum Vitae

Rusydi Hasan Makarim was born in Cilacap, Indonesia, on June 7, 1989. He grew up in the same city, where he attended public high school 1 (Sekolah Menengah Atas Negeri 1) and learned computer programming for the very first time.

In 2007 he enrolled in the undergraduate program in computer science at the International Islamic University (Universiti Islam Antarabangsa) in Kuala Lumpur, Malaysia. During his undergraduate studies, he participated in multiple ACM-ICPC programming contests at the national, regional, and international level. After his graduation in December 2011, he worked briefly as HTML5 Developer Support in ServiceRocket Sdn. Bhd. (formerly CustomWare Sdn. Bhd.) until June 2012.

In September 2012 he started the master programme in cryptography at the Institute of Applied Mathematics (Uygulamalı Matematik Enstitüsü), Middle East Technical University (Orta Doğu Teknik Üniversitesi) in Ankara, Türkiye. He wrote his master thesis with the title "Relating Undisturbed Bits to Other Properties of Substitution Boxes" under the supervision of Assoc. Prof. Dr. Ali Doğanaksoy and graduated in July 2014.

In October 2014, he started the PhD program at the Mathematical Institute, University Leiden and was also affiliated with the Cryptology Group, Centrum Wiskunde en Informatica (CWI) Amsterdam, the Netherlands. His PhD thesis is completed under the supervision of Prof.dr. Ronald Cramer and dr. Marc Stevens.

From May 2019 until March 2023, Rusydi worked as a Lead Cryptography Analyst at the Technology Innovation Institute (TII) in Abu Dhabi, United Arab Emirates. His current research work focuses primarily on the design and the cryptanalysis of symmetric-key primitives, particularly using automated tools such as Mixed-Integer Linear Program (MILP), Satisfiability Modulo Theory (SMT), and SAT solvers.

Publications

1. Emanuele Bellini, Juan Grados, Rusydi H. Makarim, Carlo Sanna. *Finding Differential Trails on ChaCha by Means of State Functions*. International Journal of Applied Cryptography, Volume 4, No. 3/4, 2024, pp. 156-175.

2. Solane El Hirch, Joan Daemen, Raghvendra Rohit, Rusydi H. Makarim. *Twin Column Parity Mixers and Gaston - A New Mixing Layer and Permutation*. 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part III, pp. 475-506.

3. Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, Thomas Peyrin. *Boosting Differential-Linear Cryptanalysis of ChaCha7 with MILP*. (2023). IACR Transactions on Symmetric Cryptology, Volume 2023, Issue 2, pp. 189-223.

4. Emanuele Bellini, David Gérault, Juan Grados, Yun Ju Huang, Rusydi H. Makarim, Mohamed Rachidi, Sharwan K. Tiwari. *CLAASP: a Crypto-*

graphic Library for the Automated Analysis of Symmetric Primitives. Selected Areas in Cryptography (SAC), Fredericton, New Brunswick, Canada, 2023, pp. 387-408.

5. Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, Thomas Peyrin. *Fully Automated Differential-Linear Attacks Against ARX Ciphers*. Cryptographers' Track at the RSA Conference (CT-RSA), San Francisco, USA, 2023, pp 252–276.

6. Stefano Barbero, Emanuele Bellini, Rusydi H. Makarim. *Rotational Analysis of ChaCha Permutation*. Advances in Mathematics of Communications, Volume 17, Issue 6, 2023, pp. 1422-1439.

7. Rusydi H. Makarim and Raghvendra Rohit. *Towards Tight Differential Bounds of Ascon: A Hybrid Usage of SMT and MILP*. (2022). IACR Transactions on Symmetric Cryptology, Volume 2022, Issue 3, pp. 303-340.

8. Emanuele Bellini, Rusydi H. Makarim, Carlo Sanna, Javier A. Verbel. *An Estimator for the Hardness of the MQ Problem*. 13th International Conference on Cryptology in Africa (AFRICACRYPT), Fez, Morocco, 2022, pp. 323-347.

9. Carlos Aguilar Melchor, Nicolas Aragon, Emanuele Bellini, Florian Caullery, Rusydi H. Makarim, Chiara Marcolla. *Constant Time Algorithms for ROLLO-I-128*. SN Computer Science, Volume 2, Number 5, September 2021, article number 382.

10. Emanuele Bellini, Alessandro De Piccoli, Rusydi H. Makarim, Sergio Polese, Lorenzo Riva, Andrea Visconti. *New Records of Pre-image Search of Reduced SHA-1 Using SAT Solvers*. Proceedings of the Seventh International Conference on Mathematics and Computing (ICMC), Shibpur, India, 2021, pp. 141-151.

11. Emanuele Bellini, Florian Caullery, Rusydi H. Makarim, Marc Manzano, Chiara Marcolla, Victor Mateu. *Advances and Challenges of Rank Metric Cryptography Implementations*. IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, 2019, pp. 325-328.

12. Putranto H. Utomo and Rusydi H. Makarim. *Solving a Binary Puzzle*. Mathematics in Computer Science, Volume 11, 2017, pp. 515-526.

13. Rusydi H. Makarim and Marc Stevens. *M4GB: An Efficient Gröbner Basis Algorithm*. Proceedings of the 2017 ACM International Symposium on Symbolic and Algebraic Computation (ISSAC), Kaiserslautern, Germany, 2017, pp. 293-300.

14. Rusydi H. Makarim and Cihangir Tezcan. *Relating Undisturbed Bits to Other Properties of Substitution Boxes*. Lightweight Cryptography for Security and Privacy - Third International Workshop (LightSec), Istanbul, Türkiye, 2014, pp. 109-125.

Courses and Summer Schools

1. Parallel Algorithms (Mastermath). Utrecht University, Utrecht, the Netherlands. Fall 2015.
2. Presentation Skills. Centrum Wiskunde en Informatica, Amsterdam, the Netherlands. Fall 2015.
3. UbiCrypt Spring School on Symmetric Cryptography. Ruhr University, Bochum, Germany. 17-19 March 2016.
4. Scientific Conduct for PhDs (Science). Leiden University (Online). 30 May 2024.

Presentations

1. *M4GB: A new Gröbner-basis Algorithm.* The 42nd International Symposium on Symbolic and Algebraic Computation (ISSAC), Technical University of Kaiserslautern, Germany. 25-28 July 2017
2. *M4GB: A new Gröbner-basis Algorithm.* ALGANT-DOC Students Meeting. Leiden University. 15 May 2017
3. *Introduction to Cryptography.* ALGANT-DOC Weekend Seminar. University of Duisburg-Essen, Germany. 27-28 February 2016.

Conferences, Seminars, and Workshops

1. Special RISC Seminar/CWI Lectures on Privacy and Security, Centrum Wiskunde en Informatica, 15 November 2018
2. RISC Seminar on Secure Multi-Party Computation and Code-Based Cryptography, Centrum Wiskunde en Informatica, 7 December 2017
3. The 8th International Workshop on Parallel Symbolic Computation (PASCO), Technical University of Kaiserslautern, 23-24 July 2017
4. The 42nd International Symposium on Symbolic and Algebraic Computation (ISSAC), Technical University of Kaiserslautern, Germany, 25-28 July 2017
5. ALGANT-DOC Students Meeting, Leiden University, 15 May 2017.
6. Mathematical Structures for Cryptography, Lorentz Center, Leiden, 22-26 August 2016
7. Directions in Symmetric Cryptography (DISC) 2016 Workshop. Bochum, Germany, 23-24 March 2016
8. The 23rd International Conference on Fast Software Encryption (FSE), Bochum, Germany, 20-23 March 2016
9. ALGANT-DOC Weekend Seminar, University of Duisburg-Essen, 27-28 February 2016
10. DIAMANT Symposium Fall 2015, Lunteren, 26-27 November 2015

11. ALGANT-DOC Students Meeting, University of Regensburg, Germany, February 2015
12. Security in Times of Surveillance, Technical University Eindhoven, 8 May 2015
13. RISC Seminar on Secret Sharing and Multiparty Computation, Centrum Wiskunde en Informatica, 24 April 2015
14. Special LACAL@RISC Seminar on Cryptologic Algorithms, Centrum Wiskunde en Informatica, 6 February 2015

Teaching Assistant

1. Cryptology (Mastermath) - Fall 2015