Matrix Algebraic Systems and Application to DAGS Cryptanalysis

Manon Bertin

Normandie Université

UNIVERSITÉ DE ROUEN NORMANDIE

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité **INFORMATIQUE**

Préparée au sein de l'**Université de Rouen Normandie**

Systèmes algébriques matriciels et application à la cryptanalyse de DAGS

Présentée et soutenue par

MANON BERTIN

Thèse soutenue le 13/12/2022

devant le jury composé de :

M. PHILIPPE GABORIT	Professeur des Universités - UNIVERSITE LIMOGES	Rapporteur du jury
M. JEAN-PIERRE TILLICH	Directeur de Recherche - CENTRE REGIONAL DE L'INRIA SACLAY ILE DE FRANCE	Rapporteur du jury
MME DELPHINE BOUCHER	Maître de Conférences HDR - UNIVERSITE RENNES 1	Membre du jury
M. ALAIN COUVREUR	Directeur de Recherche - ECOLE POLYTECHNIQUE	Membre du jury
M. OLIVIER BLAZY	Professeur des Universités - ECOLE POLYTECHNIQUE	Président du jury
M. AYOUB OTMANI	Professeur des Universités - Université de Rouen Normandie	Directeur de thèse
MME MAGALI BARDET	Maître de Conférences HDR - Université de Rouen Normandie	Co-directeur de thèse

Thèse dirigée par **AYOUB OTMANI** (LABORATOIRE D'INFORMATIQUE DE TRAITEMENT DE L'INFORMATION ET DES SYSTEMES) et **MAGALI BARDET** (LABORATOIRE D'INFORMATIQUE DE TRAITEMENT DE L'INFORMATION ET DES SYSTEMES)

MIIS

litis

Acknowledgements - Remerciements

Ce manuscrit est l'aboutissement de plusieurs années de travail, autant académique que sur moi-même. Il est évident qu'une grande partie de cette thèse a été difficile pour moi, comme elle l'est pour beaucoup. J'ai refusé d'abandonner, et ce doctorat m'a finalement permis de me connaître mieux, de me comprendre et surtout, de trouver ma voie.

Les premières personnes que je voudrais remercier sont Magali Bardet et Ayoub Otmani, qui m'ont encadrée et dirigée pendant toute cette thèse. Merci d'avoir continué à croire en moi quand j'avais moi-même du mal à le faire, d'avoir été si compréhensifs et d'avoir partagé tant de conseils.

Je tiens aussi à remercier Nicolas Forcadel et Alin Bostan pour avoir accepté de faire partie de mon Comité de Suivi de thèse. Vos conseils et remarques ont toujours été justes et m'ont permis de finir cette thèse dans de bonnes conditions.

Je souhaite vivement remercier Philippe Gaborit et Jean-Pierre Tillich pour avoir accepté d'être rapporteurs de ce manuscrit. Merci à Delphine Boucher, Olivier Blazy et Alain Couvreur pour avoir accepté de faire partie du jury pour ma soutenance.

J'ai eu l'occasion de mesurer l'importance des rencontres pendant ces quelques années, et ainsi je souhaite remercier les personnes que j'ai pu croiser, que ce soit aux journées Codes et Cryptographie, aux Journées Nationales de Calcul Formel et plus régulièrement au groupe de travail sur la cryptographie post-quantique basée sur les codes correcteurs d'erreurs. Je mesure la chance d'avoir assisté à tous ces évènements et d'y avoir tant appris.

Cette thèse s'est effectué au sein du LITIS, au croisement de l'Université de Rouen Normandie et de l'INSA de Rouen Normandie. Je voudrais donc commencer par remercier tous mes collègues du département d'Informatique. Merci particulièrement à Cécile, tu m'as énormément aidée grâce aux quelques pauses repas qui se sont transformées en longues discussions. Merci à Pascal, Bruno, Valentin et Yannick pour nos fameux "groupes de travail" du midi. On retiendra que définitivement, Race for The Galaxy n'est pas mon jeu ! Merci à Bruno pour ta réactivité dès que j'ai pu avoir des problèmes informatiques.

Je souhaiterais associer à ces remerciements les membres du département ASI de l'INSA. Après avoir été votre élève pendant quelques années, j'ai apprécié vous découvrir en tant que collègues. Merci à Cecilia, pour ta gentillesse et ton soutien notamment à l'école doctorale. Merci à Laurent, tu as marqué mes années à l'INSA et tu m'as accompagné jusqu'à mes concours cette année. Merci à Nicolas et Nicolas, Gilles, Nathalie, Clément et Samia pour leur aide.

Je remercie tout autant les collègues doctorants avec qui j'ai pu partager toutes mes années de thèses, et qui sont devenus mes amis. Merci à Amazigh, pour ton humour qui n'a jamais manqué d'animer le bureau, et pour ta profonde gentillesse. Merci à Clément, pour avoir tout autant contribué à l'animation de la salle, pour ton stock de nourriture et pour le carillon (quand il ne sonne pas trop). Merci à Etienne, pour avoir doublé le nombre de doctorant en cryptographie dans l'équipe et pour nos discussions autant académiques que personnelles. Merci à Pierre, pour m'avoir aidé à survivre au confinement, et pour le soutien que tu m'as offert pendant ces années. Merci à Marc, Younes, Samira, Safaa, Gabriel, Diane, Edwin, Elodie et Alexandre pour avoir été des compagnons de travail et, surtout, de pauses.

Toutes les démarches administratives et les préparations de missions n'auraient pas pu se faire sans l'aide des gestionnaires, secrétaires et assistant(e)s de direction. Marion, Fabienne, Solène, Brigitte, merci pour votre gentillesse et votre aide précieuse. Mathieu, merci pour ta bonne humeur et pour les innombrables longues discussions que j'ai adoré avoir avec toi.

Il y a quelques personnes sans qui cette thèse se serait finie en plein milieu et qui ont contribué à me faire reprendre pied. Je voudrais remercier le personnel du CMP de Rouen, avec une pensée toute particulière pour l'infirmière qui m'a suivie.

Je voudrais continuer ces remerciements par les personnes de ma vie qui n'étaient pas directement liées à ma thèse, mais ont contribué, à leur façon, à ce que ce manuscrit existe.

Merci aux membres de la chorale de l'INSA, pour m'avoir permis de relâcher la pression régulièrement, de m'impliquer dans les concerts et d'avoir partagé de très chouettes moments entre musiciens. Merci aux anciens du Quartier des Geeks à qui j'ai la chance de parler encore de temps en temps, j'ai beaucoup appris grâce aux projets que l'on a monté ensemble. Merci aux Archers Blancs de Pavilly, la thèse m'a empêchée d'être aussi présente que je l'aurais voulu mais votre soutien a beaucoup compté. Merci aux membres de la Bibliotèque Virtuelle et aux Alchimistes, certaines de mes séances de travail n'auraient pas été aussi productives sans vous. Merci aux collègues des établissements où j'ai enseigné depuis l'année dernière, particulièrement à Mélissa, Caroline, Sarah et Janine, merci pour votre bonne humeur et vos encouragements.

Robin, merci d'avoir été l'autre "vieux" de l'AMIR. Matthieu, merci d'être le meilleur public de mes blagues. À tous les deux, merci pour votre soutien, nos soirées et les nombreuses discussions sur nos expériences de doctorat.

Shubiao, thank you for being such a precious friend. Miao, Elodie, Thibault, Thomas, merci d'avoir pris de mes nouvelles si souvent et d'être d'immuables soutiens. Pline, Jean-Marc, Raphaël, merci de nourrir mon goût des jeux de société mais surtout, surtout, merci d'être si présents, au quotidien. Nayeem, merci d'avoir accompagné les derniers jours d'écriture de cette thèse.

Je ne pourrais finir ces remerciements sans citer ma famille, sans qui je ne serais évidemment pas là. Que ce soit mes grands-parents, oncles, tantes, cousin(e)s ou mon petit frère, vous avez tous contribué, à votre façon, à ce que j'arrive à finir ce doctorat. Maman, Papa, merci de m'avoir transmis votre amour de la culture et de la science et l'importance de ne jamais cesser d'apprendre. Je n'aurais jamais eu le courage de commencer, ni de finir cette thèse sans votre soutien.

Contents

Introduction

Cryptography has been a tool used during all eras, mostly to protect war secrets. This explains that one of the most, if not the most, famous cryptosystem is called after the emperor of Antique Roma Caesar. Encryption of messages continually improved since then, and it has been of utmost importance since the democratization of computers and of the Internet. Indeed, cryptography allows messages to be transmitted between a selected number of persons while remaining secret to everyone else. This is particularly useful to communicate safely through the Internet, or to pay with a debit card and a code.

As cryptography became a well-researched field, a need for standards arose. In the 1970s, multiple algorithms were considered, and RSA was picked against others such as McEliece. The advantage of RSA over McEliece is that it has shorter cryptographic keys that allowed to reduce the storage needed and to make the computations faster. McEliece is a code-based cryptosystem, and its security relies on the problem of decoding a linear code. As it was not chosen at that time, it was not as studied as RSA, but it came back into research when the threat of quantum computers took form. Indeed, the security of RSA does not hold again such a computer, when McEliece is, for now, still considered as a hard problem for all computers.

In 2017, the National Institute of Standards and Technology started a Call for Submissions to find the next potential algorithms that are resistant to quantum computers. By the end of the year, more than 60 cryptosystems were kept for round 1 of analysis, using different methods to do encryption and/or signature. After multiple rounds of selection, a first list of algorithms to be standardized was picked, while some other ones were chosen to be evaluated for longer. Among the algorithms of the first round was the DAGS candidate. Based on quasi-dyadic General Srivastava codes, it was unfortunately attacked during the first round and did not make it to the next one. The attack is described in the second part of this thesis, as well as improvements we made later.

Concurrently to the NIST Call for Submissions, some publications analyzed the solving of the systems built from the MinRank problem.

This solving was applied to Hamming metric as well as rank metric, improved in multiple articles. For example, the SupportMinors modeling was brought by those improvements, that also allowed a better understanding of the solving if attack on DAGS that we will present here.

Organisation of this thesis. This thesis is split in two parts, each treating a different domain, while still being related, as we use the content of one part to help with the other.

The first part goes over polynomial systems and their solving. We start by giving some definitions that create the basis to polynomial systems studying. Once the basics are covered, we detail how Gröbner bases are a very important tool to solve such systems. We then focus on a particular subfamily of polynomial systems, which are bilinear systems. We list multiples types of bilinear systems, how they were studied and some results on their complexity. Finally, we consider some recent results around bilinear systems that are expressed as a MinRank problem instance. This highlighted that the mechanism mainly involves one set of variables among the two of the system, and we noticed that we could use that to improve an algebraic attack we already worked on against the DAGS cryptosystem. This leads to the second part of this thesis.

The domain of the second part is cryptography. Indeed, we begin by talking about the basics of code-based cryptography, which is the foundation of this thesis. We define the codes and other notions that are really important after. We remind the context of the Call for Submissions from the NIST, and introduce the DAGS cryptosystem. At the same time, we describe the first attack that was published against it. The last two chapters are explaining improvements we were able to do on the attack to make it faster. We separated them in two categories : one for the changes made on inputs, like varying the number of rows of one of the matrix; and one for the changes that were more around the Gröbner basis computations.

Contributions

Solving bilinear matricial algebraic systems One of the main contributions of this thesis is the extension of [57, 3] works on matricial algebraic systems. We use the approach of multiplying the original bilinear system by vectors of minors of the jacobian of the matrix, that we can easily compute in advance. Considering the fact that those vectors of minors are elements of the kernel of the jacobian, the product induces a degree fall, and we obtain polynomials of lower degrees. In the system newly created, we consider the minors as our variables and explain how it reduces the size of the matrices we encounter in our computations, and thus the length of said computations. This process can be done for multiple degrees, and we are able to determine if the solving of the system is possible for a given set of degrees. It is described in Chapter 3.

Modelisations equivalence This analysis of the solving of bilinear matricial systems leads to another contribution. The process we just described is linked to the SupportMinor approach presented in [12]. As explained in [9], the SupportMinor modeling exists in multiple versions and this way of considering minors as variables is equivalent to one of these versions. We actually present all the different modeling and how they can be related to one another in Subsection 3.3.3

Attack improvement against DAGS This thesis also presents in Chapter 6 the way we improved the attack on the NIST candidate DAGS that was initially made by [16]. An algebraic system is built from the elements of the cryptosystem, and the private parts are recovered through the solving of this system. In order to make computations easier, the number of variables is reduced by shortening codes that are used to build the system. We remark that the ratio between the number of equations and the number of variables is actually more important than the reduction of the number of variables to its minimum. Then, starting with the work in [10] and progressing into a SupportMinor-like approach, we manage to attack all the initial sets of parameters. Concurrently, we were able to reduce the computations time so that every sets of parameters can be broken in less than a second.

Hybrid attack on DAGS updated parameters Following the attack, DAGS authors updated the parameters so that they would be secure against it. We present in Chapter 6 of this manuscript a hybrid method that allows us to attack one of these updated sets of parameters. Such a hybrid method relies on exhaustive search to go through all possible values for a subset of the variables and then, for every value tested, we try to solve the algebraic system. For some sets of parameters, we are able to estimate a complexity that is much lower than the claimed security. For others, we remark that the update makes this attack impossible.

Notation

$\boldsymbol{A}$ Matrix $\boldsymbol{A}$ (Bold uppercase letter).

$\boldsymbol{A}_{[I,J]}$ Submatrix of $\boldsymbol{A}$ where we picked only the rows whose indices are in I and the columns whose indices are in J.

$\boldsymbol{a}$ Vector $\boldsymbol{a}$ (Bold lowercase letter). We chose to have row vector by default.

$\boldsymbol{a}^T$ Transpose of the vector $\boldsymbol{a}$. Column vector.

$\boldsymbol{e}_i^n$ Canonical vector of size n with a 1 as its i-th coefficient.

$\otimes$ Kronecker product (Definition in appendix A).

$||\boldsymbol{e}||$ Hamming weight of the vector $\boldsymbol{e}$.

Part I

Solving Polynomial Systems

Chapter 1

Polynomial Systems Solving

Contents

1.1 Introduction

This thesis is at the confluence of polynomial systems solving and cryptography, and this chapter introduces the former.

Polynomial systems solving is a classical problem in mathematics, where we aim to solve a set of equations with one or more unknowns. Solving means, in the general case, that we find all the possible solutions i.e. the possible values that the unknowns can take to satisfy the equations. The number of solutions may vary depending on the space in which the values of the unknowns are and the constraints given on them. Such systems are used to model mathematical and physical problems in numerous domains, and that is why finding ways to solve them efficiently is important.

One of these domains is a big part of this thesis : cryptography. In this are of study, we create a polynomial system from the public parts of a cryptographic scheme. We then have polynomials where coefficients corresponds to public parts and unknowns to secret parts that we want to find. Solving a system means getting values for those unknowns and thus obtaining informations on parts that are not supposed to be found. We call this an *algebraic attack*. Sometimes it can even result on the knowledge of the whole secret, which is the ideal result of the attack.

We will develop about algebraic attacks later on, but first we need to define basic concepts around polynomials, polynomial systems and their solving.

Outline of the chapter

In this chapter, we begin by giving definitions about monomials, polynomials and polynomial systems in section 1.2, including those of ideals and varieties that are essential for the whole thesis. This first section allows us to tackle the solving of those polynomial systems in section 1.3, particularly with Gröbner bases as detailed in subsection 1.3.2 and their complexity in section 1.4. Finally, we finish this chapter with the definitions and properties of regular and semi-regular sequences in 1.5.

1.2 Polynomial Systems

In this very first section we introduce definitions about polynomials and systems of polynomials, as well as ideals and varieties. All those elements are essential as they are used throughout this thesis.

The definitions and propositions presented here correspond to those from [25] and [24], unless otherwise specified.

1.2.1 Definitions

We consider $x_1, \dots, x_n$ some variables.

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

$$x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$$

where each degree α_i of a variable x_i is a non-negative integer. The sum $\alpha_1 + \alpha_2 + \cdots + \alpha_n$ is the total degree *of a monomial, that we sometimes abbreviate as x^α with α the list of the degrees α_i.*

Definition 1.2. *Let $\mathbb{K}$ be a field. A* polynomial *is a finite linear combinations of monomials with coefficients in $\mathbb{K}$. We can write a polynomial in $x_1, \dots, x_n$ as :*

$$f = \sum_{\alpha} c_\alpha x^\alpha$$

where c_α are coefficients in $\mathbb{K}^n$. For a given α, we will refer to $c_\alpha x^\alpha$ as a term.

We denote by $\mathbb{K}[x_1, \dots, x_n]$ the ring of all the polynomials in $x_1, \dots, x_n$ with coefficients in $\mathbb{K}$.

When studying cryptosystems, as we will do later on, we rarely encounter a polynomial by itself. Most of the time, we see them as a set of polynomials that we aim to solve and that we call polynomial system.

Definition 1.3. *Let $f_1, \dots, f_m$ a set of polynomials in $\mathbb{K}[x_1, \dots, x_n]$. A* polynomial system *is the collection of those polynomials.*

Another way of writing this is:

$$\begin{array}{c} f_1(x_1, \dots, x_n) \\ f_2(x_1, \dots, x_n) \\ \vdots \\ f_m(x_1, \dots, x_n) \end{array}$$

We will sometimes write such a polynomial system as $\mathcal{F}$.

Our goal being to solve a polynomial system, we will search for the potential values of $x_1, \dots, x_n$ to obtain the results $\mathcal{F} = 0$.

Depending on their properties, we can give a name to some specific systems. We present here some systems whose names come from the degrees of the polynomials in their equations and we will highlight why we are interested in them.

Definition 1.4. *A polynomial system is said to be* linear *when the maximum degree of a monomial in each equation is* 1*. Therefore, we can write a linear system as:*

$$\boldsymbol{A}\boldsymbol{x}^T - \boldsymbol{b}$$

where $\boldsymbol{A}$ *is the matrix of the coefficients,* $\boldsymbol{x}$ *is the vector containing the variables and* $\boldsymbol{b}$ *is the vector containing the constant parts.*

Considering that it is always possible to solve linear systems in a polynomial time, they are often easier to solve than other systems: we will give details in section 1.3.

Definition 1.5. *A polynomial is said to be* quadratic *when the maximum degree among its monomials is* 2*. We can write a quadratic polynomial as:*

$$\boldsymbol{x}\boldsymbol{A}\boldsymbol{x}^T + \boldsymbol{b}\boldsymbol{x} - c$$

where $\boldsymbol{A}$ *and* $\boldsymbol{b}$ *are respectively a matrix and a vector of coefficients,* $\boldsymbol{x}$ *is the vector containing the variables, and* c *is a constant.*

In this thesis we will be particularly interested in a subfamily of quadratic systems: bilinear systems. The maximal degree of a bilinear system is also 2, but the system is using two sets of variables and is linear in each one of these sets. We will not give more details about these systems here, as we will discuss them in depth in chapter 2.

Another property that will be discussed in this thesis is the homogeneity of a system. Therefore, we introduce the notion of homogeneous systems :

Definition 1.6. *A polynomial is said to be* homogeneous *if all its monomials with nonzero coefficients have the same total degree. We can extend this definition to polynomial systems : they are said to be* homogeneous *when all their polynomials are homogeneous.*

Homogeneous systems are important because they are used as references for computing the complexity of algorithms made for solving systems. Unfortunately, we encounter many systems that are not homogeneous, named *affine systems*. When dealing with such systems, we divide them into two parts : the homogeneous part of highest degree and the affine part which is of lower degree. We focus the analysis on the homogeneous part of highest degree as we can predict the behavior of the algorithms and then examine the effects on the affine part. Again, these will be discussed further in section 1.3 and chapter 2.

1.2.2 Ideals and Varieties

Now that we tackled the basics about polynomial systems that are going to be useful for this thesis, we also need to define the particular subsets of rings named ideals. To do so, we have to introduce polynomial combinations first.

Definition 1.7. *Let $f_1, \dots, f_m$ polynomials from $\mathbb{K}[x_1, \dots, x_n]$. We denote by $\langle f_1, \dots, f_n \rangle$ the polynomial combination:*

$$\langle f_1, \dots, f_m \rangle = \left\{ \sum_{i=1}^{m} p_i f_i : p_i \in \mathbb{K}[x_1, \dots, x_n] \text{ for } i = 1, \dots, m \right\}$$

Definition 1.8. *Let $I \subset \mathbb{K}[x_1, \dots, x_n]$ a non-empty subset. I is said to be an* ideal *if*

- $\forall f \in I, \forall g \in I, f + g \in I$
- $\forall f \in I, \forall p \in \mathbb{K}[x_1, \dots, x_n], pf \in I$

From Definitions 1.7 and 1.8, we can deduce that $\langle f_1, \dots, f_m \rangle$ is an ideal *generated* by $f_1, \dots, f_m$. We say that we have an *homogeneous ideal* when there exists a system of homogeneous generators for this ideal. Moreover, if an ideal has a finite number of solutions, its *degree* corresponds to this number of solutions, taking into account the potentiel multiplicities.

Lastly, we need to define an object corresponding to the set of solutions of the system we want to solve.

Definition 1.9. *Let $\mathbb{K}$ be a field and $f_1, \dots, f_m$ be polynomials in $\mathbb{K}[x_1, \dots, x_n]$. We define the* variety *$V(f_1, \dots, f_m)$ as:*

$$V(f_1, \dots, f_m) = \{(a_1, \dots, a_n) \in \mathbb{K}^n : f_i(a_1, \dots, a_n) = 0, \forall i \in [1, \dots, m]\}$$

In other words, it corresponds to the set of all solutions in $\mathbb{K}$ of the system:

$$\begin{cases} f_1(x_1, \dots, x_n) & = 0 \\ f_2(x_1, \dots, x_n) & = 0 \\ & \vdots \\ f_m(x_1, \dots, x_n) & = 0 \end{cases}$$

The set $\mathbb{K}$ can be either an infinite field or, as we will see mainly in this thesis, a finite field. However, using this definition in an infinite field such as $\mathbb{R}$ or $\mathbb{C}$ is hard, contrary to applying it on a finite field. Indeed, working on a finite field implies that we have algebraic constraints that are easier to deal with. When searching for a variety in a finite field $\mathbb{F}_q$, we can add multiple *field equations* of the form $x_i^q - x_i = 0$. This assures us that the computation will give a set a solutions of the polynomial system in the right space.

1.3 Solving Polynomial Systems

Polynomial systems are meant to be solved, that is to say we want to find values of the variables $x_1, \dots, x_n$ such that all the equations are true at the same time. Depending on the properties of the systems, the meaning of solving may differ.

In the case of a positive dimensional system, i.e. a system with an infinite number of solutions, it is impossible to enumerate each one of them. Thus we aim for a general formula or property applicable to all of them. In the case of a zero-dimensional system, i.e. a system with a finite number of solutions, solving means finding all those solutions. This is actually the case of most of the systems that we will study in this thesis. Moreover, in cryptography, we do not necessarily want to find all the solutions, but to recover a sufficient amount of information about secret parts of a scheme. Finding one solution among the set is sometimes enough to get relevant information.

The difficulty of finding a solution depends on the degrees of the polynomials, the number of variables and the number of polynomials. In some cases, such as those we encounter in cryptography, some structure may be added to the system, impacting its difficulty. Generically, a system with more variables than equations is said to be *underdetermined*, and a system with more equations than variables is said to be *overdetermined*. If the number of equations surpasses greatly the number of variables we sometimes use the word *superdetermined*. In this section, we will tackle the solving of generic polynomial systems. Some special cases will be discussed later in chapter 2.

1.3.1 Monomial orders

The division for polynomials in one variable is known. A legitimate question is: does a similar notion exist for multivariate polynomials? It actually exists, and to introduce it, we will first define what is a monomial order and present some orders that will be useful later on.

Definition 1.10. *A* monomial order *on* $\mathbb{K}[x_1, \ldots, x_n]$ *is any relation* $>$ *on the set of monomials of the form* x^α *in* $\mathbb{K}[x_1, \ldots, x_n]$ *that has the following properties :*

- $>$ *is a* total ordering *relation: any two distinct elements can be comparable, and this relation is transitive.*
- $>$ *is* compatible with multiplication *in* $\mathbb{K}[x_1, \ldots, x_n]$ *: for* x^α, x^β *and* x^γ *monomials in* $\mathbb{K}[x_1, \ldots, x_n]$, *if* $x^\alpha > x^\beta$ *then* $x^\alpha x^\gamma > x^\beta x^\gamma$.
- $>$ *is a* well-ordering*:* $\forall x^\alpha \in \mathbb{K}[x_1, \ldots, x_n], x^\alpha > 1$.

Example 1.11. There exists only one monomial order for univariate monomials. It depends solely on the degree of the monomials:

$$\cdots > x^{n+1} > x^n > \cdots > x^3 > x^2 > x > 1$$

When we consider multivariate monomials, there are much more possibilities for monomial orders. First, when writing them as $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$, we implicitly set up the following order :

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

But even then, there are a lot of possibilities for ordering multivariate monomials. We present here two monomial orders that we will use regularly in this thesis. To do so, let $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ be two sequences of degrees.

Definition 1.12. *The* lexicographic order (lex) *is similar to the way words are ordered in a dictionary:*

$$x^\alpha >_{\text{lex}} x^\beta \text{ if } \exists i \text{ such that } \begin{cases} \alpha_j = \beta_j & \text{for } j < i \\ \alpha_i > \beta_i \end{cases}$$

This order is the most intuitive one, and it is the best to use for solving a system. Indeed, the variables are separated into distinct groups, which makes linear algebra computations easier by allowing to eliminate one variable after one variable. Let us see an example of the application of this order on 3-variables monomials to illustrate it better:

Example 1.13. For the variables $x_1 >_{\text{lex}} x_2 >_{\text{lex}} x_3$, the lexicographic ordering of the monomials of degree up to 3 is $x_1^3 > x_1^2x_2 > x_1^2x_3 > x_1^2 > x_1x_2^2 > x_1x_2x_3 > x_1x_2 > x_1x_3^2 > x_1x_3 > x_1 > x_2^3 > x_2^2x_3 > x_2^2 > x_2x_3^2 > x_2x_3 > x_2 > x_3^3 > x_3^2 > x_3 > 1$.

Definition 1.14. *The* graded reverse lexicographic order (grevlex) *is the order such that:*

$$x^\alpha >_{\text{grevlex}} x^\beta \text{ if } \begin{cases} \alpha_1 + \cdots + \alpha_n > \beta_1 + \cdots + \beta_n \\ \textit{or} \\ \alpha_1 + \cdots + \alpha_n = \beta_1 + \cdots + \beta_n \\ \textit{and} \begin{cases} \alpha_j = \beta_j & \text{pour } j > i \\ \alpha_i < \beta_i \end{cases} \end{cases}$$

This grevlex order will particularly interests us because it is the easiest to work with when computing Gröbner bases as we will see in subsection 1.3.2. As we did with lex order, we provide an example of the application of the grevlex order on 3-variables monomials:

Example 1.15. For the variables $x_1 >_{\text{grevlex}} x_2 >_{\text{grevlex}} x_3$, the graded reverse lexicographic ordering of the monomials of degree up to 3 is $x_1^3 > x_1^2x_2 > x_1x_2^2 > x_2^3 > x_1^2x_3 > x_1x_2x_3 > x_2^2x_3 > x_1x_3^2 > x_2x_3^2 > x_3^3 > x_1^2 > x_1x_2 > x_2^2 > x_1x_3 > x_2x_3 > x_3^2 > x_1 > x_2 > x_3 > 1$

The two orders we presented here are used for different purposes. It may seems overcomplicated to make an ordering change when dealing with a different step of the solving of the system.

Actually, there exists an algorithm called FGLM [32], that allows this change to be done for ideals with a finite number of solutions. Its complexity is well-estimated: it is similar to a linear algebra computation. With n variables, I an

ideal and $D = \deg(I)$ its degree, the FGLM algorithm complexity is bounded by $O\left(nD^3\right)$. Using the optimized order for each step and using the FGLM algorithm in between makes for a lower complexity than using only one order for the full solving process.

Now that we defined some orders, we can decide on a way to write polynomials. In the following, we will consider polynomials as sum of monomials, ordered from the highest to the lowest according to the order chosen. This leads to the following definitions:

Definition 1.16. *Let us consider $f = \sum_{\alpha} c_\alpha x^\alpha$ a polynomial and $>$ an ordering. The* leading term *is the product $c_\alpha x^\alpha$ where x^α is the highest monomial in f according to the ordering $>$ for which $c_\alpha \neq 0$. Then, we call c_α the* leading coefficient *and x^α the* leading monomial. *We will denote them respectively $\mathrm{LT}_>(f)$, $\mathrm{LC}_>(f)$ and $\mathrm{LM}_>(f)$.*

When the order that we use is clear or if any order can be used, we will not specify it. We can now introduce the method we will use to solve polynomial systems.

1.3.2 Gröbner Bases

The principal method to solve polynomial systems we will discuss in this thesis is using Gröbner Bases. In order to define such bases, we need first to extend the definition of univariate polynomials division to multivariate polynomials:

Definition 1.17 (Multivariate Polynomial Division)**.** *Let $\mathcal{F} = (f_1, \ldots, f_m)$ be an ordered tuple of polynomials in $\mathbb{K}[x_1, \ldots, x_n]$. Given a monomial order $>$, we can write every $f \in \mathbb{K}[x_1, \ldots, x_n]$ in the form of :*

$$f = a_1 f_1 + \cdots + a_m f_m + r$$

where $a_i, r \in \mathbb{K}[x_1, \ldots, x_n]$ such that either $r = 0$ or r is a linear combination of monomials. r is called a remainder *of f on division by $\mathcal{F}$, and its computation is called a* reduction*. We can also write r as $\overline{f}^{\mathcal{F}}$. If r is a linear combination of monomials that are not divisible by any of the leading terms $\mathrm{LT}_>(f_1), \ldots, \mathrm{LT}_>(f_m)$, then this is a* total reduction*.*

This definition allows us to divide a polynomial f by m polynomials $f_1, \ldots, f_m$ that we will consider generators of some ideal I. Also, we can have different a_i and r depending on the monomial order $>$ and the order of the polynomials f_i.

We now have all the tools needed to define Gröbner Bases:

Definition 1.18. *Let $>$ be a monomial order on $\mathbb{K}[x_1, \ldots, x_n]$ and $I \subset \mathbb{K}[x_1, \ldots, x_n]$ be an ideal. A* Gröbner basis *for I with respect to $>$ is a finite collection $\mathcal{G} = \{g_1, \ldots, g_t\} \subset I$ such that, for every nonzero $f \in I$, $\mathrm{LT}_>(f)$ is divisible by $\mathrm{LT}_>(g_i)$ for some i. The Gröbner basis $\mathcal{G}$ is a basis of I.*

An important thing with reducing with Gröbner basis is that given an order $>$, the remainder will always be the same, independently of the order of the g_i.

There exists multiple algorithms to compute Gröbner bases. The historical one is the Buchberger algorithm [22], which go through every possible couple of polynomials of an ideal to find new elements of the basis. However, other algorithms emerged with another way of doing the computations. Indeed, Lazard [45] showed that it is possible to compute Gröbner bases by applying linear algebra to the Macaulay matrix. More recently were introduced the algorithms F_4 in [29] and F_5 in [30] that are based on the Buchberger mechanisms but rely on linear algebra on the Macaulay matrix.

The complexity of computing a Gröbner basis is a very important point that we will address later on.

1.4 Complexity Tools

It is one thing to know a method, it is another to actually be able to apply it. Obstacles to do so can be that the computations take too long or use too much memory on a computer. To avoid these kinds of situation and be able to find an actual solution to a system, we can study the complexity of the solving. Here we will present tools that help to do this.

1.4.1 Definitions

We begin by introducing a notation that will allow us to refer to all polynomials of degree $\leq s$ or exactly s.

Definition 1.19. *Let s be an integer, and I be an ideal over $\mathbb{K}[x_1, \ldots, x_n]$. We denote all the polynomials of degree $\leq s$ of $\mathbb{K}[x_1, \ldots, x_n]$ as:*

$$\mathbb{K}[x_1, \ldots, x_n]_{\leq s}$$

We can apply that to an ideal:

$$I_{\leq s} = I \cap \mathbb{K}[x_1, \ldots, x_n]_{\leq s}$$

We obtain $I_{\leq s}$, which is a vector subspace of $\mathbb{K}[x_1, \ldots, x_n]_{\leq s}$.

We can use a similar definition to restrict our space to polynomials of degree exactly s and for I an homogeneous ideal. Then, we denote $\mathbb{K}[x_1, \ldots, x_n]_s$ the polynomials of $\mathbb{K}[x_1, \ldots, x_n]$ of degree s and $I_s = I \cap \mathbb{K}[x_1, \ldots, x_n]_s$ the polynomials of degree s in I.

Associated with this separation of polynomials by degree, we have the Macaulay matrix, which is a matricial representation of the system. It was introduced by Macaulay in [46].

Definition 1.20. *The* Macaulay matrix *of degree d of a system of polynomials $(f_1, \ldots, f_m)$, homogeneous in the variables $(x_1, \ldots, x_n)$, is the matrix that*

describes every polynomials of the form $u \cdot f_i$ with u a monomial, such that $\deg(u \cdot f_i) = d$, in function of monomials of degree d. More precisely, it is a matrix where each column is indexed by monomials of degree d and each row is indexed by the products $u \cdot f_i$. Each element of the matrix corresponds to the coefficient of the monomial of the column in the polynomial product of the row.

Let us see an example of a Macaulay matrix:

Example 1.21. Let $f_1 = x_1^3 + x_1x_2x_3$, $f_2 = x_1^2 - x_1x_3 + x_2x_3$ and $f_3 = x_1x_2^2 - x_3^3$ be three homogeneous polynomials in variables (x_1, x_2, x_3). We want to write the corresponding Macaulay matrix of degree 3. The polynomials f_1 and f_3 are already of degree 3, so we use them directly. However, the polynomial f_2 is only of degree 2, so we will have to consider all products $u \cdot f_2$ such that u takes the value of all possible monomials of degree $3 - 2 = 1$. It gives us three new polynomials that will be new rows of the matrix:

$$\begin{aligned} x_1 \cdot f_2 &= x_1^3 - x_1^2x_3 + x_1x_2x_3 \\ x_2 \cdot f_2 &= x_1^2x_2 - x_1x_2x_3 + x_2^2x_3 \\ x_3 \cdot f_2 &= x_1^2x_3 - x_1x_3^2 + x_2x_3^2 \end{aligned}$$

Finally, we get the following Macaulay matrix:

$$\begin{array}{c} \\ f_1 \\ x_1 \cdot f_2 \\ x_2 \cdot f_2 \\ x_3 \cdot f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccccccccc} x_1^3 & x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^2x_3 & x_1x_2x_3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \end{array}$$

1.4.2 Hilbert functions and series

Thanks to the tools previously defined, we can now introduce Hilbert functions [39, 40].

Definition 1.22. *Let I be an ideal in $\mathbb{K}[x_1, \ldots, x_n]$, and $s \in \mathbb{N}$. For homogeneous systems, the* Hilbert function *is the function defined by:*

$$\begin{aligned} \mathrm{HF}_I(s) &= \dim\left(\mathbb{K}[x_1, \ldots, x_n]_s / I_s\right) \\ &= \dim\left(\mathbb{K}[x_1, \ldots, x_n]_s\right) - \dim(I_s) \end{aligned}$$

Property 1.23. *[39, 24] Let I be an ideal, and z a formal variable. The* Hilbert series *is the formal series which coefficients are given by the Hilbert function:*

$$\mathrm{HS}(z) = \sum_{s \geq 0} \mathrm{HF}_I(s)\, z^s$$

It corresponds to the following rational fraction:

$$\frac{P(z)}{(1-z)^d}$$

with $P(1) \neq 0$. Thus d is the dimension of I and $P(1)$ the degree of the variety defined by I.

Because they illustrate a relation between the dimension of the ideal and the degree of the variety, Hilbert series are a useful tool to compute the complexity of solving a given system.

Hilbert series are only relevant for homogeneous systems, that is why we chose here to not define them for affine systems. However, for homogeneous systems of dimension zero, that is to say with a finite number of solutions, there is a unique solution which is $\mathbf{0}$. When dealing with affine systems, we use the Hilbert series of the homogeneous part of highest degree as an indication for the complete system complexity in the generic case.

1.4.3 Complexity computations

The goal here when computing Hilbert series is to get a complexity formula for applying an echelon form to a Macaulay matrix. This would allow us to also have a formula for the computation of the associated Gröbner basis, as their link was highlighted in [45].

We consider a system of polynomials $(f_1, \ldots, f_m)$ in the variables $x_1, \ldots, x_n$. For all $i \in \{1, \ldots, m\}$, we denote d_i the degree of the polynomial f_i. Let $\mathbb{K}[x_1, \ldots, x_n]_d$ be the ring of homogeneous polynomials of degree d, and I_d the intersection $\mathbb{K}[x_1, \ldots, x_n]_d \cap I$, which represents the homogeneous ideal of degree d.

Definition 1.24. *The complexity of getting the echelon form of a matrix of size $R_d \times C_d$ of rank $rank_d$ can be bounded [43, 1] by*

$$O\left(C_d \times R_d \times rank_d^{\omega-2}\right)$$

with ω the constant that express the complexity of the linear algebra used in this computation. The current best bound for ω is 2.37286 as stated in [2].

To get a formula with elements that are known, we compute all the following elements:

- the number of columns C_d: as we want to apply the formula of complexity on a Macaulay matrix, it corresponds to the number of monomials of degree d in the variables $x_1, \ldots, x_n$. This is given by the following binomial coefficient:
$$C_d = \binom{n+d-1}{d}$$
- the number of rows R_d: for a Macaulay matrix, it corresponds to the number of monomials of degree $d - d_i$ for each polynomial f_i. This gives us the following sum:
$$R_d = \sum_{i=0}^{m} \binom{n+d-d_i-1}{d-d_i}$$

- the rank: the rank of the Macaulay matrix can be computed from the formula of the Hilbert series. Indeed, from the Hilbert series, we have:

$$\begin{aligned}\mathrm{HF}(d) &= \dim\left(\mathbb{K}[x_1,\ldots,x_n]_d / I_d\right)\\ &= \dim\left(\mathbb{K}[x_1,\ldots,x_n]_d\right) - \dim\left(I_d\right)\\ &= \dim\left(\mathbb{K}[x_1,\ldots,x_n]_d\right) - rank_d\end{aligned}$$

  As the dimension of $\mathbb{K}[x_1,\ldots,x_n]_d$ corresponds to the number of monomials of degree d, we have:

$$rank_d = \binom{n+d-1}{d} - \mathrm{HF}(d)$$

Now that we have all the elements that were not explicit, we obtain the final formula for all the Macaulay matrices up to a degree D, with F a constant that depends only on the matrix.

$$\sum_{d=0}^{D}\left(\binom{n+d-1}{d} \times \sum_{i=0}^{m}\binom{n+d-d_i-1}{d-d_i} \times \left(\binom{n+d-1}{d} - \mathrm{HF}(d)\right)^{\omega-2} \times F\right)$$

We can give an upper bound to this formula, that we can find in [14]:

$$O\left(mD\binom{n+D}{D}^{\omega}\right)$$

1.5 Regular and Semi-regular Sequences

In this section we introduce regular and semi-regular sequences, their respective properties and their associated Hilbert series.

1.5.1 Regular Sequences

Regular sequences were introduced by Macaulay [46, 47]. Their behavior can be predictable and corresponds to the *average behavior* of polynomial sequences. We remind that $\mathbb{K}$ a field and $\mathbb{K}[x_1,\ldots,x_n]$ the polynomial ring with n variables with coefficients in $\mathbb{K}$.

Definition 1.25. *A sequence $f_1,\ldots,f_m$ of m homogeneous polynomials in $\mathbb{K}[x_1,\ldots,x_n]$ is said to be* regular *if the following statements are verified:*

- $\langle f_1,\ldots,f_m\rangle \neq \mathbb{K}[x_1,\ldots,x_n]$
- $\forall i \in [1;m]$, *if* $g_i f_i = 0$ *in* $\mathbb{K}[x_1,\ldots,x_n]/\langle f_1,\ldots,f_{i-1}\rangle$ *then* $g_i = 0$ *in* $\mathbb{K}[x_1,\ldots,x_n]/\langle f_1,\ldots,f_{i-1}\rangle$

Now that we have introduced the notion, we can give some properties of these sequences:

Proposition 1.26. *Let $f_1, \ldots, f_m$ be a homogeneous regular sequence, and d_i the degree of f_i. Then the following properties are verified:*

1. *the ideal $\langle f_1, \ldots, f_m \rangle$ has a dimension $n - m$*
2. *the Hilbert series of the polynomial sequences is:*
$$\sum_{d \geq 0} \mathrm{HF}_d(n) z^d = \prod_{i=1}^{m} \frac{(1 - z^{d_i})}{(1 - z)^n}$$
3. *any permutation $f_{\sigma(1)}, \ldots, f_{\sigma(m)}$ is also a regular sequence*
4. *the index of regularity is the Macaulay bound $\sum_{i=1}^{m} (d_i - 1) + 1$*
5. *almost every sequence is a regular sequence : the set of sequences of n variables and degrees $d_1, \ldots, d_m$ that are regular is a Zariski open set.*

Proof. The first property appears in [24]. Proof of the second one can be found in [24] as well as in [8]. Property 3 is proven in [28] and property 4 in [45]. Lastly, the proof of property 5 is in [8]. □

1.5.2 Semi-regular Sequences

Semi-regular sequences are an extension of regular sequences for the overdetermined case. Most of the definitions and the properties given here are from [8], and for all of them we consider the ideals to be homogeneous and zero-dimensional. Unless otherwise specified, all the proofs can be found in [8] as well.

We begin by giving a definition of semi-regular sequences, similar to the one we previously presented for regular sequences:

Definition 1.27. *Let $f_1, \ldots, f_m \in \mathbb{K}[x_1, \ldots, x_n]$ be a homogeneous sequence. It is said to be* semi-regular *if the following conditions are verified:*

- $\langle f_1, \ldots, f_m \rangle \neq \mathbb{K}[x_1, \ldots, x_n]$
- $\forall i \in [1; m]$, *if* $g_i f_i = 0$ *in* $\mathbb{K}[x_1, \ldots, x_n] / \langle f_1, \ldots, f_{i-1} \rangle$ *and* $\deg(g_i f_i) < H(I)$ *then* $g_i = 0$ *in* $\mathbb{K}[x_1, \ldots, x_n] / \langle f_1, \ldots, f_{i-1} \rangle$

Similarly to the regular sequences, we can now give some properties of semi-regular sequences.

Proposition 1.28. *Let $f_1, \ldots, f_m$ be a homogeneous semi-regular sequence, d_i the degree of f_i and $I = \langle f_1, \ldots, f_m \rangle$. Then the following properties are verified:*

1. *the sequence $f_1, \ldots, f_m$ is semi-regular in $\mathbb{K}[x_1, \ldots, x_n]$ if and only if its Hilbert series is:*
$$HS_I(z) = \left[\prod_{i=1}^{m} \frac{(1 - z^{d_i})}{(1 - z)^n} \right]$$

with $\left[\prod_{i=1}^{m} \frac{(1-z^{d_i})}{(1-z)^n}\right] = \sum_{d\geq 0} h_{d,m}(n) z^d$ *the* generating series *of the sequence* $f_1, \ldots, f_m$. *The notation with brackets means that we keep only the terms of the sequence whose coefficients are strictly positive.*

2. *any permutation* $f_{\sigma(1)}, \ldots, f_{\sigma(m)}$ *is also a semi-regular sequence*

3. *when the variety associated to the ideal is zero-dimensional (i.e.* $m \geq n$*), the index of regularity* $H(I)$ *is characterized by:*

$$\forall d < H(I), \quad h_{d,m}(n) > 0 \quad \text{and} \quad h_{H(I)}(n) \leq 0$$

4. $f_1, \ldots, f_m$ *being semi-regular does not imply that for all* $i \in [1; m]$*, the sequence* $f_1, \ldots, f_i$ *is semi-regular.*

When the sequence of polynomials is in $\mathbb{K}[x_1, \ldots, x_n]$, we can determine equivalence between some of those previous properties:

Proposition 1.29. *For a given sequence of* $m = n$ *polynomials of* $\mathbb{K}[x_1, \ldots, x_n]$*, the following properties are equivalent:*

- *the sequence is regular*
- *the sequence is semi-regular*
- *its Hilbert series is*

$$\prod_{i=1}^{m} \frac{(1-z^{d_i})}{(1-z)^n}$$

Chapter 2

Bilinear Systems

Contents

2.1 Introduction

We presented in the previous chapter polynomials as well as polynomial systems, linear and quadratic. They both have interesting properties, and we will focus on a family mixing properties of both: bilinear systems. These systems have a maximum degree of 2, and are linear in each set of variables they are using. As we quickly mentioned before, there exist homogeneous systems where all monomials have the same degree d and affine systems where d is the maximum degree while there is a part of lower degree.

Bilinear systems appear in various scientific domains to model different situations. We will not detail those possibilities, but focus on the one that interest us particularly here: cryptography. Indeed, in order to execute an algebraic cryptographic attack, it happens that the system we build is bilinear. We then need to study the complexity of solving the obtained system, to know if we are able to go through all the computations successfully. It occurs sometimes that the structure of the system is even more particular: while still being bilinear, it can be written as a multiplication of some matrices.

This chapter discusses about all these possibilities.

Outline of the chapter

In this chapter, we will first define bilinear systems in section 2.2, insisting on the two possible distribution of degrees: homogeneous in subsection 2.2.1 and affine in subsection 2.2.2. In section 2.3, we address the solving of such systems and the complexities of some given systems previously studied. Finally, on the last section 2.4, we will consider a special case of bilinear systems where those are expressed as a product of matrices.

2.2 Bilinear Systems

Bilinear systems are a subfamily of quadratic systems. Indeed, all monomials are of degree 2 but we have two separate groups of variables that both have a maximum degree equal to 1.

We consider $\mathbb{F}$ to be a field of characteristic 2 : although the analysis we will present here can be applied to fields with a different characteristic, we will keep 2 as it allows us to neglect signs in formulas. Moreover, it is consistent with the systems we will create in the attack of DAGS in part II.

For this section we will use $R = \mathbb{F}\left[x_1, \ldots, x_{n_x}, y_1, \ldots, y_{n_y}\right]$ a polynomial ring with two blocks of variables. Thus, the systems we will observe here are systems of polynomial equations bilinear in $\boldsymbol{x} = (x_1, \ldots, x_{n_x})$ and $\boldsymbol{y} = \left(y_1, \ldots, y_{n_y}\right)$.

Among those systems, we can have two possibilities : either they are homogeneous, and we only have monomials of degree 2 in the polynomials, or they can be affine, and we can have monomials of lower degrees. Each case will have its own section, here we begin by homogeneous bilinear systems.

2.2.1 Homogeneous bilinear systems

We begin by detailing the bilinear systems that are homogeneous. They are important because we are able to predict the generic complexity of solving them much more easily than for affine systems, and when dealing with affine systems, we use homogeneous systems as models for complexity. Let us begin by defining a bilinear equation:

Definition 2.1. *Let $\boldsymbol{M}$ be a matrix of size $n_x \times n_y$ with coefficients in $\mathbb{F}$. A* bilinear equation *in $\boldsymbol{x}$ and $\boldsymbol{y}$ can be written as the following product:*

$$f : \boldsymbol{x}\boldsymbol{M}\boldsymbol{y}^T = \sum_{\ell=1}^{n_x} x_\ell \boldsymbol{M}_{[\ell,*]}\boldsymbol{y}^T = 0 \tag{2.1}$$

We rarely deal with one bilinear equation only, but with multiple ones organized as a bilinear system, that can be defined as:

Definition 2.2. *A* bilinear system $\mathcal{F} = (f_1, \ldots, f_m) \in R$*, expressed as a column vector of m equations, can be written as*

$$\mathcal{F} : \sum_{\ell=1}^{n_x} x_\ell \boldsymbol{M}_\ell \boldsymbol{y}^T = 0 \text{ where } \boldsymbol{M}_\ell = \begin{pmatrix} \boldsymbol{M}^{(1)}_{[\ell,*]} \\ \boldsymbol{M}^{(2)}_{[\ell,*]} \\ \vdots \\ \boldsymbol{M}^{(m)}_{[\ell,*]} \end{pmatrix} \tag{2.2}$$

with each matrix $\boldsymbol{M}^{(i)}$ being associated to the equation f_i. The matrices $\boldsymbol{M}_\ell$ are of size $m \times n_y$.

For the analysis we will do later on, we need two other definitions, the first one being about syzygies:

Definition 2.3. *A* syzygy *of $\mathcal{F} = (f_1, \dots, f_m) \in R$ is a sequence of polynomials $(g_1, \dots, g_m) \in R$ such that:*

$$\sum_{i=1}^{m} g_i f_i = 0 \tag{2.3}$$

Remark 2.4. It is important to distinguish equations where the goal is to find a solution by finding potential values for unknowns, and polynomial identities to 0 such as in the previous definition where the equality with zero is a constraint for the sequence of polynomials.

Syzygies are not strictly restricted to bilinear systems and equations, they can be used with any category of polynomials. We can say the same thing for the jacobian matrix, although with bilinear systems, these matrices have a particularity. We present that in the next definition:

Definition 2.5. *We can define the* jacobian matrix *of the system $\mathcal{F}$ with respect to $\boldsymbol{x}$ as*

$$\mathrm{jac}_{\boldsymbol{x}}(\mathcal{F}) = \left(\frac{\partial f_i}{\partial x_j}\right)_{\substack{1 \le i \le m \\ 1 \le j \le n_x}} \tag{2.4}$$

Likewise, we can define $\mathrm{jac}_{\boldsymbol{y}}(\mathcal{F})$ the jacobian matrix of $\mathcal{F}$ with respect to $\boldsymbol{y}$. If $\mathcal{F}$ is a bilinear system, the entries of these jacobian matrices are particular. They are linear polynomials: $\mathrm{jac}_{\boldsymbol{x}}(\mathcal{F})$ has entries linear in $\boldsymbol{y}$ and $\mathrm{jac}_{\boldsymbol{y}}(\mathcal{F})$ has entries linear in $\boldsymbol{x}$.

We can see it with the jacobian matrices for the system given in (2.2):

$$\mathrm{jac}_{\boldsymbol{x}}(\mathcal{F}) = \left(\boldsymbol{M}_1 \boldsymbol{y}^T, \dots, \boldsymbol{M}_{n_x} \boldsymbol{y}^T\right) \tag{2.5}$$

$$\mathrm{jac}_{\boldsymbol{y}}(\mathcal{F}) = \sum_{l=1}^{n} x_l \boldsymbol{M}_l \tag{2.6}$$

It is obvious here that each jacobian here has entries that are respectively linear in $\boldsymbol{y}$ and $\boldsymbol{x}$. This allows us to give the following lemma:

Lemma 2.6. *Considering $\mathcal{F}$ a bilinear system, we have :*

$$\mathrm{jac}_{\boldsymbol{x}}(\mathcal{F}) \cdot \boldsymbol{x}^T = \mathcal{F}^T \tag{2.7}$$

It implies that finding an element from the left kernel of the jacobian cancels the homogeneous system, i.e. we find syzygies of $\mathcal{F}$ that are dependent on $\boldsymbol{y}$.

We have seen definitions and properties of homogeneous bilinear systems and we will now introduce similar notions about affine bilinear systems.

2.2.2 Affine bilinear systems

Let's now consider f an affine bilinear polynomial in $\boldsymbol{x}$ and $\boldsymbol{y}$. With the same notation as before, we can write it as :

$$\boldsymbol{x}\boldsymbol{M}\boldsymbol{y}^T + \boldsymbol{x}\boldsymbol{a}^T + \boldsymbol{b}\boldsymbol{y}^T + c = (\boldsymbol{x}, 1)\begin{pmatrix}\boldsymbol{M} & \boldsymbol{a}^T \\ \boldsymbol{b} & c\end{pmatrix}\begin{pmatrix}\boldsymbol{y}^T \\ 1\end{pmatrix} \tag{2.8}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors of respective lengths n_x and n_y, and c is a constant.

It can be divided into 2 parts : the homogeneous part of highest degree $f_h = \boldsymbol{x}\boldsymbol{M}\boldsymbol{y}^T$ and the affine part $f_a = \boldsymbol{x}\boldsymbol{a}^T + \boldsymbol{b}\boldsymbol{y}^T + c$. We can notice that f_h, considered alone, is actually a homogeneous bilinear polynomial and all the definitions and properties presented before can apply to it. We can extend those definitions to systems, with $\mathcal{F}_h$ the homogeneous part of highest degree of the complete system and $\mathcal{F}_a$ the affine part of the system.

During the solving of such systems, we use products by polynomials to create new ones. Those newly created polynomials have a degree that is at most the sum of the degree of previous polynomials. If the degree we obtain is lower, we can say that a degree fall happened.

Definition 2.7. *Let $\mathcal{F} = (f_1, f_2, \ldots, f_m)$ a system of polynomials with an homogeneous part of degree d, and $(g_1, g_2, \ldots, g_m)$ a set of polynomials. If the polynomial resulting from the sum $\sum_{i=1}^{m} f_i g_i$ has a degree lower than d, we say that the computations induced a* degree fall.

The computations with the new polynomials are faster if they are of lower degrees, thus having degree falls helps. For affine bilinear systems, we can induce a degree fall by extending Lemma 2.6:

Lemma 2.8. *For $\mathcal{F}$ an affine bilinear system, we can write:*

$$\mathrm{jac}_{\boldsymbol{x}}(\mathcal{F}_h) \cdot \boldsymbol{x}^T + \mathcal{F}_a^T = \mathcal{F}_h^T + \mathcal{F}_a^T \tag{2.9}$$

Let $\boldsymbol{v}$ be an element from the left kernel of $\mathrm{jac}_{\boldsymbol{x}}(\mathcal{F}_h)$, thus this element cancels the homogeneous part of the system. We obtain :

$$\boldsymbol{v} \cdot \mathcal{F}_a^T = \boldsymbol{v}\left(\mathcal{F}_h^T + \mathcal{F}_a^T\right) = \boldsymbol{v} \cdot \mathcal{F}^T \tag{2.10}$$

The new equations obtained with $\boldsymbol{v} \cdot \mathcal{F}_a^T$ belong to the ideal generated by $\mathcal{F}$. By definition, $\mathcal{F}_h$ is of a higher degree than $\mathcal{F}_a$, its cancellation induces a degree fall.

This theorem is interesting when $\boldsymbol{v}$ is not a syzygy of the affine part, which is the case in general.

2.3 Solving Bilinear Systems

Now that we have seen what are homogeneous and affine bilinear systems, we will present how their respective properties are influencing the complexity of

solving them. We consider [24] as our main reference unless otherwise specified. We will begin by introducing some useful definitions and explaining some basic notions about complexity, and then we will give some theoretical formulas for generic homogeneous and affine bilinear systems.

2.3.1 Definitions around complexity

We use Gröbner basis with respect to the grevlex order, which we already presented as the best monomial order for this.

Homogeneous Bilinear Systems Let's now dive into the first category of systems we want to talk about here: homogeneous bilinear systems. Something to notice for these systems is that the Macaulay matrices that are computed at each step of the Gröbner basis computations are indexed by the monomials of degree exactly d for columns and by the polynomials mf_i such that $\deg(mf_i) = d$ for the rows. The complexity of homogeneous bilinear systems is easier to analyse than the one for affine bilinear systems, as we know that the biggest part of it corresponds to the cost of computing a row echelon form on all the Macaulay matrices up to degree d, d being the largest polynomial degree encountered. A tool that we can use to help evaluate complexity of a Gröbner basis computation, is the *degree of regularity* of a system.

Definition 2.9. *Let $\mathcal{F}$ be a homogeneous zero-dimensional system, its* degree of regularity *corresponds to:*

$$d_{\text{reg}}(\mathcal{F}) = \min\left\{d \in \mathbb{Z}^+ \,|\, \dim(I_d) = \dim(R_d)\right\}$$

with R the polynomial ring $\mathbb{F}[\boldsymbol{x}]$.

This degree of regularity has the benefit of depending only on the ideal generated by the system. It is related to the complexity of solving a system, making it interesting to study. For some families its value has been precisely estimated, we will give some examples in the subsection 2.3.2.

Affine Bilinear Systems Let us consider now the case of affine systems, which is different. First, the Macaulay matrix is indexed by the monomials of degree $\leq d$ for columns and by the polynomials mf_i such that $\deg(mf_i) \leq d$ for the rows. However, it is not linked to the complexity of solving the system anymore, due to the presence of partial syzygies, i.e. syzygies of the homogeneous parts only that induce degree falls, as explained in lemma 2.8. In order to estimate the complexity, we need to introduce the notion of first degree fall:

Definition 2.10. *Let $\mathcal{F}$ be a system that we want to solve using a Gröbner basis computation. The* first degree fall*, that we will denote d_{ff}, is the smallest degree d of the Gröbner basis computation such that there is a degree fall in d on $\mathcal{F}$.*

We consider that it is representative of the complexity of solving the system when the computation ends shortly after reaching d_{ff}. If this is not the case, a higher degree is reached later during the computation and the complexity is not related anymore to the value of d_{ff}.

2.3.2 Known Complexities for some Specific Systems

It is too difficult to estimate a precise complexity for every bilinear system. However, some precise families of systems were studied throughout the years and we present here the estimations that resulted.

Affine Overdetermined Systems with Semi-Regular Homogeneous Leading Part These systems were studied in 2005 in [13]. This article studied affine systems such that the homogeneous leading part is semi-regular. Thus, we can use the following d_{max} as a good measure of complexity.

$$d_{max} \leq \deg\left(\left[\frac{(1-z^2)^{\#eq}}{(1-z)^{\#vars}}\right]\right) + 1$$

Generic Bilinear Affine Square Systems Those were explored in 2011 in [31]. Before giving the complexity result of this article, we will talk about another result that will be very important for the rest of the thesis.

Let us consider $\boldsymbol{x} = (x_1, \ldots, x_{n_x})$ and $\boldsymbol{y} = (y_1, \ldots, y_{n_y})$ two vectors of unknowns, and $\mathcal{F} = (f_1, \ldots, f_m)$ a set of polynomials such that $m = n_x + n_y$. A system $\mathcal{F} = 0$ respecting the previous constraints is said to be a *square system*.

We want to understand the algebraic structure of such an affine bilinear system, and for that we need to study the two following jacobian matrices:

$$\text{jac}_{\boldsymbol{x}}(\mathcal{F}) = \left(\frac{\partial f_i}{\partial x_j}\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,n_x}} \quad \text{and} \quad \text{jac}_{\boldsymbol{y}}(\mathcal{F}) = \left(\frac{\partial f_i}{\partial y_j}\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,n_y}}$$

We remind that in the case of bilinear systems, the jacobian matrices coefficients are only linear forms. The elements of the kernels of those matrices cancel the homogeneous part of the system, which is interesting as it can induce a degree fall. According to Conjecture 1 from [31], all elements in these kernels are vectors of maximal minors of the jacobian matrices.

Let us give an example for a small system.

Example 2.11. We can consider a system with $n_x = 3$ and $n_y = 3$ unknowns for $m = 4$ polynomials. Then, we have the following elements in the respective kernels of $\text{jac}_{\boldsymbol{x}}(\mathcal{F})$ and $\text{jac}_{\boldsymbol{y}}(\mathcal{F})$:

$$u = \left(\text{minor}\left(\text{jac}_{\boldsymbol{x}}(\mathcal{F}), 1\right), -\text{minor}\left(\text{jac}_{\boldsymbol{x}}(\mathcal{F}), 2\right), \text{minor}\left(\text{jac}_{\boldsymbol{x}}(\mathcal{F}), 3\right), -\text{minor}\left(\text{jac}_{\boldsymbol{x}}(\mathcal{F}), 4\right)\right)$$
$$v = \left(\text{minor}\left(\text{jac}_{\boldsymbol{y}}(\mathcal{F}), 1\right), -\text{minor}\left(\text{jac}_{\boldsymbol{y}}(\mathcal{F}), 2\right), \text{minor}\left(\text{jac}_{\boldsymbol{y}}(\mathcal{F}), 3\right), -\text{minor}\left(\text{jac}_{\boldsymbol{y}}(\mathcal{F}), 4\right)\right)$$

where minor $(\text{jac}_{\boldsymbol{x}}(\mathcal{F}), i)$ is the determinant of the matrix obtained by removing the i-th row from $\text{jac}_{\boldsymbol{x}}(\mathcal{F})$. The two vectors u and v are generic syzygies that cancel the homogeneous part of the system, provoking a degree fall.

Now that we presented this result about maximal minors, we can give the upper bound on the degree of regularity for this kind of "square" systems :

$$d_{\text{reg}} \leq \min(n_x, n_y) + 2$$

This is given for a "square" system, however if we have an overdetermined system, we can get a similar estimation by removing enough equations to get back to a square system. The estimation will correspond to an upper bound for our system, as adding new equations, i.e. new constraints, will help with decreasing the complexity of solving the system.

We shared here the estimates of two families of bilinear systems, but the systems that we have for our attacks are even more precise: they are written as products of matrices. We will see in the next section that it has been recently studied a lot and that we give a closer estimation than the ones introduced here.

2.4 MinRank Instances

We consider now a particular case of a bilinear system, which is a modeling allowing to solve the MinRank problem. We will first present the problem and the associated modeling, and we will explain how this relates to the systems we previously introduced.

2.4.1 MinRank problem and system

The MinRank problem was first introduced in [33]. The problem was not named MinRank yet, but Gabidulin was the first to formulate it. It is an NP-complete problem [23], and it can be stated as follows :

Definition 2.12. *Let r be a positive integer and $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_m$ matrices of size $p \times n$ over $\mathbb{F}$. Find $x_1, \ldots, x_m$, not all zero, such that:*

$$\text{Rank}\left(\sum_{i=1}^{m} x_i \boldsymbol{M}_i\right) \leq r \tag{2.11}$$

In [44], Kipnis and Shamir introduced a new way to present the problem, as a system to be solved. We will call it the *KS method*, where we solve the MinRank problem by finding $x_1, \ldots, x_m, k_1, \ldots, k_{r(n-r)} \in \mathbb{F}$ such that :

$$\left(\sum_{i=1}^{m} x_i \boldsymbol{M}_i\right)\begin{pmatrix}\boldsymbol{I}_{n-r} \\ \boldsymbol{K}\end{pmatrix} = 0 \quad \text{where} \quad \boldsymbol{K} = \begin{pmatrix} k_1 & \cdots & k_{r(n-r-1)+1} \\ \vdots & \ddots & \vdots \\ k_r & \cdots & k_{r(n-r)} \end{pmatrix} \tag{KS}$$

We can notice that the system described here is bilinear in two sets of variables $x_1, \ldots, x_m$ and $k_1, \ldots, k_{r(n-r)}$. The homogeneous part of highest degree has monomials of bidegree $(1,1)$, while the affine part has monomials of bidegree $(0,1)$, $(1,0)$ and $(0,0)$.

The MinRank problem and its derivatives have been recently studied profusely in [57, 3, 12, 11]. Each one of these articles relates improved ways to solve the systems created in order to break the corresponding cryptosystem. We will present these improvements in the chapter 6, focusing on what will be useful for our attack on DAGS.

Remark 2.13. It is to be noticed that we can apply the methods we will present when the systems we are studying have a form that is a transpose of KS, that is to say:

$$\left(\boldsymbol{I}_{n-r} \quad \boldsymbol{U}\right)\left(\sum_{i=1}^{m} x_i \boldsymbol{M}_i'\right) = 0 \quad \text{with} \quad \boldsymbol{U} = \boldsymbol{K}^T \text{ and } \boldsymbol{M}_i' = (\boldsymbol{M}_i)^T$$

2.4.2 Matricial Bilinear Systems

We can notice in the Kipnis-Shamir modeling for the MinRank problem that it actually is a bilinear system were the second block of variables is not a vector $\boldsymbol{y}$ but a $\nu \times n_y$ matrix $\boldsymbol{Y}$. If we consider the homogeneous part of highest degree, we can write it as:

$$\mathcal{F}_h : \sum_{l=1}^{n_x} x_l \boldsymbol{M}_l \boldsymbol{Y}^T = 0 \tag{2.12}$$

It is the apparition of the matrix $\boldsymbol{Y}$ that generates the block structure. Each block of the system uses the same coefficients and variables from $\boldsymbol{x}$ but only a specific subset of variables from $\boldsymbol{Y}$. Such a block, that we will denote $\mathcal{F}^{(i)}$, can be expressed with a formula similar to (2.2) :

$$\mathcal{F}_h^{(i)} : \sum_{l=1}^{n_x} x_l \boldsymbol{M}_l \left(\boldsymbol{Y}_{[i,*]}\right)^T = 0 \qquad \text{with } i = \{1, \ldots, \nu\} \tag{2.13}$$

Remark 2.14. It is important to notice that the whole system generated is a matrix of equations, thus it is necessary to be cautious about the order in which we take the equations and the variables when doing an analysis.

We can now consider the affine system, where we add the part of lower degrees to the homogeneous part of highest degree. We can notice that there is an affine part in the modeling of the MinRank problem, due to the identity part of one of the matrices. We can write a generic affine bilinear system as:

$$\boldsymbol{x}\boldsymbol{M}\boldsymbol{Y}^T + \boldsymbol{x}\boldsymbol{A} + \boldsymbol{b}\boldsymbol{Y}^T + \boldsymbol{c} = (\boldsymbol{x}, 1)\begin{pmatrix} \boldsymbol{M} & \boldsymbol{A} \\ \boldsymbol{b} & \boldsymbol{c} \end{pmatrix}\begin{pmatrix} \boldsymbol{Y}^T \\ \boldsymbol{I}_\nu \end{pmatrix} \tag{2.14}$$

where $\boldsymbol{A}$ is a matrix of size $n_x \times \nu$, $\boldsymbol{b}$ and $\boldsymbol{c}$ are vectors of respective lengths n_y and ν and $\boldsymbol{I}_\nu$ is the identity matrix of size ν.

The homogeneous part of highest degree is $\mathcal{F}_h = \boldsymbol{x}\boldsymbol{M}\boldsymbol{Y}^T$ and the affine part is $\mathcal{F}_a = \boldsymbol{x}\ \boldsymbol{A} + \boldsymbol{b}\ \boldsymbol{Y}^T + \boldsymbol{c}$. The result of lemma 2.8 still holds for such systems which means that we can try to cancel the homogeneous part of highest degree and thus inducing a degree fall.

Chapter 3

Solving Superdetermined MinRank Instances

Contents

3.1 Introduction

We previously saw what a matricial bilinear system looks like. We remind the general formula:

$$\mathcal{F}: \quad \underbrace{\sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)} \boldsymbol{U}^T}_{\mathcal{F}_h} \quad + \underbrace{\sum_{\ell=1}^{m} x_\ell \boldsymbol{A}^{(\ell)} + \boldsymbol{B}\boldsymbol{U}^T + \boldsymbol{C}}_{\mathcal{F}_a} = 0$$

with $\boldsymbol{B}$ a matrix of size $n \times n_y$ and $\boldsymbol{C}$ a matrix of size $n \times \nu$.

We want to solve such a system, and we will explore multiple connected ways to do so. We recall the results from [57], and explain how we can compute syzygies of the system that induce a degree fall in the system. To do so, we create a new matrix, and the syzygies are expressed with the elements of the kernel of this matrix.

We will need a new notation for blocks of polynomials that use a subset of variables of the whole system containing all the x variables and only variables from the row i of the matrix $\boldsymbol{U}$:

$$\mathcal{F}_h^{(i)}: \quad \sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)} \boldsymbol{U}_{[i,*]}^T$$

Finally, we will study the Macaulay matrix of the system as a whole, for all possible degrees. We notice that its structure is particular and linked to the previous approach, as we see blocks with a structure similar to the matrix of the first method presented. We will also compare this process with SupportMinors, another modeling of similar systems which was presented in 2020 [12].

Outline of the chapter

In this chapter, we will begin by using the results from [57] on matricial bilinear systems in Section 3.2. Then in 3.3 we will explore another method which uses the Macaulay matrix of the whole system and will compare it to SupportMinors.

3.2 Superdetermined MinRank Instances

Let us consider now that we have a MinRank-like instance with a system written as :

$$\begin{pmatrix} \boldsymbol{I}_{n-r} & \boldsymbol{U} \end{pmatrix} \left(\sum_{\ell=1}^{m} x_\ell \boldsymbol{M}_\ell \right) = 0$$

with $\boldsymbol{U}$ a matrix of unknowns of size $(n-r) \times r$ and $\boldsymbol{x}$ a vector of unknowns of size m. It is clear that the system is bilinear and as we can see, it is expressed as a product of two matrices.

The problem here is how we can solve it: we can use the studies presented in subsection 2.3.2 of the previous chapter, but $\boldsymbol{U}$ being a matrix instead of a vector induces changes in the structure of the system. Solving such system has been studied in [57], and we will remind those results here.

We separate the two sets of variables in the analysis: even if the goal of finding elements of the kernel of the jacobian matrix is common, the ways to do so are not. We will begin by the study of the jacobian with linear entries in the x variables and we will study the other one after.

3.2.1 x variables

Let us focus first on the x variables. We begin by computing the jacobian matrix of the homogeneous part of highest degree in variables from $\boldsymbol{U}$, whose entries are linear in x. The jacobian matrix here has a very specific form: due to the structure of the system, there is a repetition of the same submatrix. It happens for each block $\mathcal{F}_h^{(i)}$, that is to say for each set of polynomials that uses the same set of u variables. The following lemma explains this repetition:

Lemma 3.1. *For all $i \in \{1, \ldots, n-r\}$, we have:*

$$\mathrm{jac}_{\boldsymbol{U}_{[1,*]}} \left(\mathcal{F}_h^{(1)} \right) = \mathrm{jac}_{\boldsymbol{U}_{[i,*]}} \left(\mathcal{F}_h^{(i)} \right) \tag{3.1}$$

We also have:

$$\mathrm{jac}_{\boldsymbol{U}_{[i,*]}} \left(\mathcal{F}_h^{(j)} \right) = 0 \text{ for } j \neq i \tag{3.2}$$

by definition of the $\mathcal{F}_h^{(i)}$ blocks that only contain a subset of the u variables.

Proof. We remind the equation for a block of variables of the system :

$$\mathcal{F}_h^{(i)} : \sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)} \left(\boldsymbol{U}_{[i,*]} \right)^T = 0$$

From this, we can deduce the jacobian matrix for a block $\mathcal{F}_h^{(i)}$:

$$\mathrm{jac}_{\boldsymbol{U}_{[i,*]}} \left(\mathcal{F}_h^{(i)} \right) = \sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)}$$

We can notice that the matrix in the right part does not actually depend on the value of i. Thus, we can deduce that for $i \in \{1, \dots, n-r\}$, we have

$$\text{jac}_{\boldsymbol{U}_{[1,*]}}\left(\mathcal{F}_h^{(1)}\right) = \text{jac}_{\boldsymbol{U}_{[i,*]}}\left(\mathcal{F}_h^{(i)}\right)$$

□

From this lemma we know the form of every non-zero block of the jacobian matrix. We can deduce a formula for the complete jacobian of the homogeneous part of the highest degree. To do so we need to define the vectorization:

Definition 3.2. *The vectorization of a matrix $\boldsymbol{A}$ of size $n_A \times m_A$ is the vertical vector of size $n_A m_A$ containing all the coefficients of A in a chosen order. The* row vectorization*, denoted $\text{vec}_{row}(\boldsymbol{A})$, corresponds to the concatenation of successive rows. Similarly, the* column vectorization*, denoted $\text{vec}_{col}(\boldsymbol{A})$, corresponds to the concatenation of successive columns.*

We can now use the vectorization to express the jacobian matrix expression:

Lemma 3.3.

$$\text{jac}_{\text{vec}_{row}(\boldsymbol{U})}\left(\text{vec}_{col}\left(\mathcal{F}_h\right)\right) = \boldsymbol{I}_{n-r} \otimes \left(\sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)}\right) \tag{3.3}$$

Proof. By definition of the jacobian, we can write:

$$\begin{aligned}
&\text{jac}_{\text{vec}_{row}(\boldsymbol{U})}\left(\text{vec}_{col}\left(\mathcal{F}_h\right)\right) \\
&= \left[\text{jac}_{\boldsymbol{U}_{[1,*]}}\left(\text{vec}_{col}\left(\mathcal{F}_h\right)\right) | \dots | \text{jac}_{\boldsymbol{U}_{[n-r,*]}}\left(\text{vec}_{col}\left(\mathcal{F}_h\right)\right)\right]
\end{aligned}$$

The block structure implies that variables from $\boldsymbol{U}_{[i,*]}$ are only in block $\mathcal{F}_h^{(i)}$:

$$= \left[\left(\boldsymbol{e}_1^{n-r}\right)^T \otimes \text{jac}_{\boldsymbol{U}_{[1,*]}}\left(\mathcal{F}_h^{(1)}\right) | \dots | \left(\boldsymbol{e}_{n-r}^{n-r}\right)^T \otimes \text{jac}_{\boldsymbol{U}_{[n-r,*]}}\left(\mathcal{F}_h^{(n-r)}\right)\right]$$

Thanks to Lemma 3.1, we can write all the blocks the same way :

$$\begin{aligned}
&= \left[\left(\boldsymbol{e}_1^{n-r}\right)^T \otimes \text{jac}_{\boldsymbol{U}_{[1,*]}}\left(\mathcal{F}_h^{(1)}\right) | \dots | \left(\boldsymbol{e}_{n-r}^{n-r}\right)^T \otimes \text{jac}_{\boldsymbol{U}_{[1,*]}}\left(\mathcal{F}_h^{(1)}\right)\right] \\
&= \left[\left(\boldsymbol{e}_1^{n-r}\right)^T | \dots | \left(\boldsymbol{e}_{n-r}^{n-r}\right)^T\right] \otimes \text{jac}_{\boldsymbol{U}_{[1,*]}}\left(\mathcal{F}_h^{(1)}\right) \\
&= \boldsymbol{I}_{n-r} \otimes \text{jac}_{\boldsymbol{U}_{[1,*]}}\left(\mathcal{F}_h^{(1)}\right) \\
&= \boldsymbol{I}_{n-r} \otimes \left(\sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)}\right)
\end{aligned}$$

□

Now that we know the complete jacobian matrix, we need to find the elements of its kernel so we can apply Lemma 2.8. Due to the form of this matrix that uses a Kronecker product with an identity matrix, we only need to find the elements of the left kernel of $\sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)}$. Assuming that Conjecture 1 from [31] is true, this kernel is a set of vectors of minors.

Definition 3.4. *Let $\boldsymbol{A}$ be a matrix of size $p \times r$, and J be a set such that $J \subset \{1, \dots, p\}$ of size $r+1$. We define the vector of minors $\boldsymbol{v}_J(\boldsymbol{A})$:*

$$\left(\boldsymbol{v}_J(\boldsymbol{A})\right)_j = \begin{cases} \left|(\boldsymbol{A})_{J\setminus\{j\},*}\right| & \text{if } j \in J \\ 0 & \text{if } j \notin J \end{cases} \tag{3.4}$$

We can directly apply this definition for $\boldsymbol{A} = \sum_{\ell=1}^{m} x_\ell \boldsymbol{M}^{(\ell)}$. There are $\binom{p}{r+1}$ elements in this kernel and each nonzero component of an element of the kernel is a minor of size r. As each entry of $\boldsymbol{v}_J$ is a homogeneous linear polynomial in the $\boldsymbol{x}$ variables, we expect to find elements of degree r in the kernel.

Thus, as stated in Theorem 1 of [57], with high probability the first fall degree will happen at the degree $r+2$. Indeed, during the computations, we are multiplying the syzygies of degree r we found thanks to the process we just presented by the homogeneous bilinear polynomials of degree 2 that we had beforehand. This actually results in the cancellation of the homogeneous part at the degree $r+2$, giving us new polynomials of degree at most $r+1$.

Remark 3.5. Most of the time, this analysis around the x variables does not provide the most interesting results compared to what we will present in the next subsection. However, in some cases where the next subsection does not lead to good enough syzygies, it can be interesting to check with the x variables.

3.2.2 u variables

We now focus on the u variables, that is to say the variables that represent the unknowns in the matrix $\boldsymbol{U}$. As for x, we begin by computing the jacobian matrix of the homogeneous part of highest degree in variables from $\boldsymbol{x}$, whose entries are linear in u. This case will be more complicated than the previous one due to the different distribution of the variables. Indeed, for the x variables, they were structured in blocks so the jacobian matrix was composed of a repeated block which made it easier to compute. For the u variables, all the variables are possibly found in all the equations, which makes it harder.

3.2.2.1 The matrix B

To get a jacobian matrix with a structure that will be useful for our future computations, we need the following lemma :

Lemma 3.6. *For all matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, we have :*

$$\mathrm{vec}_{row}\left(\boldsymbol{ABC}\right) = \left(\boldsymbol{A} \otimes \boldsymbol{C}^T\right) \mathrm{vec}_{row}\left(\boldsymbol{B}\right) \tag{3.5}$$

The proof of this lemma is given in Appendix A.

We can apply this lemma to the jacobian matrix we have, and we obtain the following proposition:

Proposition 3.7.

$$\mathrm{jac}_{\boldsymbol{x}}\left(\mathrm{vec}_{row}\left(\mathcal{F}_h\right)\right) = \left(\mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\boldsymbol{U}^T\right)\right)_{\ell=\{1\ldots m\}} \tag{3.6}$$

$$= \left(\boldsymbol{I}_m \otimes \boldsymbol{U}\right)\left(\mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\right)\right)_{\ell=\{1\ldots m\}} \tag{3.7}$$

As for the x variables, we want to find the kernel of the jacobian matrix. This time we need more work, and we need to introduce some new objects that will be useful later on.

Let $d \in \mathbb{N}^*$ the degree in which we want to do the computations. We define three sets of indices :

$$\begin{aligned} J &= (j_1, \ldots, j_{d+1}) && \text{such that } 1 \leq j_k < j_{k+1} \leq n-r \\ T &= (t_1, \ldots, t_d) && \text{such that } 1 \leq t_k < t_{k+1} \leq r \\ T' &= \left(t'_1, \ldots, t'_{d+1}\right) && \text{such that } 1 \leq t'_k < t'_{k+1} \leq r \end{aligned}$$

We define a block matrix $\boldsymbol{B}$, whose blocks are indexed by the different values of T for the rows and T' for the columns. Thus, there are $\binom{r}{d}$ rows and $\binom{r}{d+1}$ columns, as they are the respective number of possible values for T and T'. Each block is a zero matrix, unless T' corresponds to the union of T with exactly one element of $\{1, \ldots, r\}$ that we will denote s. The whole block is then dependent on this s, and we can have the same block with different T and T' as long as their difference is s. Thus, we define a block as follows:

$$\boldsymbol{B}_{T,T'} = \begin{cases} \boldsymbol{B}_s = \left(\boldsymbol{M}^{(\ell)}_{[*,s]}\right)_{\ell=\{1\ldots m\}} & \text{if } T' \backslash T = \{s\} \\ 0_{n \times m} & \text{else} \end{cases}$$

This matrix $\boldsymbol{B}$ is going to be our main tool to find the kernel of the jacobian matrix. The following theorem explains how we will be able to use it:

Theorem 3.8 (Theorem 2 from [57])**.** *Let $\mathbb{F}$ be a field, d be an integer such that $0 < d+1 < \min\{n-r, r\}$, $J = (j_1, \ldots, j_{d+1})$ a set of indices such that $1 \leq j_k < j_{k+1} \leq n-r$ and $\boldsymbol{a}_{T_1}, \boldsymbol{a}_{T_2}, \ldots, \boldsymbol{a}_{T_\ell} \in \mathbb{F}^n$.*

$$\left(\boldsymbol{a}_{T_1}, \boldsymbol{a}_{T_2}, \ldots, \boldsymbol{a}_{T_\ell}\right) \in \ker\left(\boldsymbol{B}\right) \text{ iff } \sum_T \boldsymbol{a}_T \otimes \boldsymbol{v}_J \in \ker\left(\mathrm{jac}_{\boldsymbol{x}}\left(\mathrm{vec}_{row}\left(\mathcal{F}_h\right)\right)\right)$$

Proof. Let J and T be two sets of indices as defined previously. We use the vector $\boldsymbol{v}_J$, which is similar to the one defined in (3.4) but with smaller minors, and can notice that, with $T' = T \cup \{s\}$:

$$\boldsymbol{v}_J \cdot \boldsymbol{U} = \sum_{s \notin T} \det\left(\boldsymbol{U}_{[J,T']}\right).\boldsymbol{e}_s^r$$

We finally define another vector :

$$V_J = \sum_T \boldsymbol{a}_T \otimes \boldsymbol{v}_J$$

We want to find elements from the kernel of the Jacobian matrix :

$$\begin{aligned} V_J \cdot \mathrm{jac}_{\boldsymbol{x}}\left(\mathrm{vec}_{row}\left(\mathcal{F}_h\right)\right) &= \sum_T \left(\boldsymbol{a}_T \otimes \boldsymbol{v}_J\right)\left(\boldsymbol{I}_m \otimes \boldsymbol{U}\right)\left(\mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\right)\right)_{\ell=\{1\ldots m\}} \\ &= \sum_T \left(\boldsymbol{a}_T \otimes \boldsymbol{v}_J\boldsymbol{U}\right)\left(\mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\right)\right)_{\ell=\{1\ldots m\}} \\ &= \sum_T \left(\boldsymbol{a}_T \otimes \sum_{s \notin T} \det\left(\boldsymbol{U}_{[J,T']}\right)\boldsymbol{e}_s^r\right)\left(\mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\right)\right)_{\ell=\{1\ldots m\}} \end{aligned}$$

with $T' = T \cup \{s\}$

$$\begin{aligned} &= \sum_T \sum_{s \notin T} \det\left(\boldsymbol{U}_{[J,T']}\right)\left(\boldsymbol{a}_T \otimes \boldsymbol{e}_s^r\right)\left(\mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\right)\right)_{\ell=\{1\ldots m\}} \\ &= \sum_T \sum_{s \notin T} \det\left(\boldsymbol{U}_{[J,T']}\right)\boldsymbol{a}_T\left(\boldsymbol{M}^{(\ell)}_{[*,s]}\right)_{\ell=\{1\ldots m\}} \\ &= \sum_{T'} \det\left(\boldsymbol{U}_{[J,T']}\right) \sum_{s \in T'} \boldsymbol{a}_{T'\setminus\{s\}}\left(\boldsymbol{M}^{(\ell)}_{[*,s]}\right)_{\ell=\{1\ldots m\}} \end{aligned}$$

We can write, for a specific T' :

$$\begin{aligned} \sum_{s \in T'} \boldsymbol{a}_{T'\setminus\{s\}}\left(\boldsymbol{M}^{(l)}_{[*,s]}\right)_{l=\{1\ldots m\}} &= \sum_{s \in T'} \boldsymbol{a}_{T'\setminus\{s\}}\boldsymbol{B}_s \\ &= \sum_T \boldsymbol{a}_T \boldsymbol{B}_{T,T'} \end{aligned}$$

The blocks of $\boldsymbol{B}$ are the $\boldsymbol{B}_{T,T'}$ matrices we introduced before, thus the size of $\boldsymbol{B}$ is $\binom{r}{d}n \times \binom{r}{d+1}m$. We can find an element that cancels all $\sum_T \boldsymbol{a}_T\boldsymbol{B}_{T,T'}$ if and only if we can construct an element of the kernel of the jacobian matrix. Indeed, the polynomial gets canceled if and only if the coefficients of all its monomials are zero: here, all monomials are actually different minors, so the only way to cancel it is to have zero coefficients. □

Now we know how we can find the kernel of the jacobian matrix, and thus syzygies for our system. We can use the following conjecture about the rank of the matrix $\boldsymbol{B}$:

Conjecture 3.9 ([57], Conjecture 1)**.** *Suppose* $\binom{r}{d}m > \binom{r}{d+1}n$*, d such that* $d+1 \leq \min\{m-r, r\}$*,* $n \leq mr$ *and* $J = (j_1, \ldots, j_{d+1})$ *such that* $1 \leq j_k < j_{k+1} \leq n-r$*.*

If the matrices $\boldsymbol{M}^{(\ell)}$*, with* $\ell \in \{1 \ldots n\}$ *are chosen uniformly at random, then with overwhelming probability in the size of* $\mathbb{F}$*, we have :*

$$Rank(\boldsymbol{B}) = \binom{r}{d+1}n \tag{3.8}$$

3.2.2.2 Remaining System

We found how to express the elements of the kernel of the jacobian matrix and the syzygies. We then obtain a new system with the equations corresponding to the affine part multiplied by the syzygies and we have new variables. Let us count equations and variables in this new system.

Corollary 3.10 ([57], Corollary 1)**.** *Suppose* $\binom{r}{d}m > \binom{r}{d+1}n$*, d such that* $d+1 \leq \min\{m-r, r\}$*,* $n \leq mr$*,* $J = (j_1, \ldots, j_{d+1})$ *such that* $1 \leq j_k < j_{k+1} \leq n-r$*, and Conjecture 3.9 holds.*

If the matrices $\boldsymbol{M}^{(\ell)}$*, with* $\ell \in \{1 \ldots n\}$ *are chosen uniformly at random, then with overwhelming probability, there is a set of* $\binom{r}{d}m - \binom{r}{d+1}n$ *syzygies of* $\mathcal{F}_h$ *of degree d.*

This corollary gives us the number of syzygies that we can expect to get from this method, and thus how many polynomials of a lower degree we can get at the degree of first fall.

3.2.2.3 Complexity

The complexity of solving such a system, if the computations finish soon after the first fall, depends on the following degree :

$$d_{min} = \min\left\{d \mid \left[\binom{r}{d}m > \binom{r}{d+1}n\right], 1 \leq d \leq r-1\right\} \tag{3.9}$$

When we reach this degree d and multiply the polynomials we have with the syzygies we computed, we have a degree fall from $d+2$ to $d+1$. Then the complexity of applying this method, given in [57], is:

$$O\left(\binom{r\kappa + d_{min} + 1}{d_{min} + 2}^{\omega}\right) = O\left(\binom{r\kappa + D - 1}{D}^{\omega}\right) = O\left((r\kappa)^{D\omega}\right)$$

with κ the number of rows of variables from $\boldsymbol{U}$ that we keep, ω the linear algebra complexity constant and $D = d_{min} + 2$.

3.3 Changing Variables Using Minors

From the previous analysis, we can see that minors are everywhere. Every time we compute new polynomials in order to have a degree fall, we actually see

minors of multiple degrees appearing. A legitimate question is then : what happens if we consider minors as variables in the new polynomials ?

3.3.1 Associated Macaulay Matrix and Rank

We want to build the Macaulay matrix of a system where we consider the minors as new variables and all the possible degrees. The advantage of such a matrix is that we can just multiply it by a vector containing all the developed minors and we can go back to the initial system. Moreover, the sizes of the matrices are then reduced by a lot, as we will see in the results.

3.3.1.1 Building the Matrix

We noticed before the omnipresence of minors in our analysis, and with [3], we can say that the degree falls appear from the product of the initial system by minors in the u variables. Let us recall this initial system, as well as the sets of indices J, T and T':

$$\begin{pmatrix} \boldsymbol{I}_{n-r} & \boldsymbol{U} \end{pmatrix} \left(\sum_{i=1}^{m} x_i \boldsymbol{M}_i \right) = 0$$

$$\begin{aligned} J &= (j_1, \ldots, j_{d+1}) && \text{such that } 1 \leq j_k < j_{k+1} \leq n-r \\ T &= (t_1, \ldots, t_d) && \text{such that } 1 \leq t_k < t_{k+1} \leq r \\ T' &= \left(t'_1, \ldots, t'_{d+1}\right) && \text{such that } 1 \leq t'_k < t'_{k+1} \leq r \end{aligned}$$

We multiply the system by a vector $\boldsymbol{v}_{J,T}$ of minors of $\boldsymbol{U}$ of degree d, as we defined previously.

$$\begin{aligned} &\boldsymbol{v}_{J,T} \cdot \begin{pmatrix} \boldsymbol{I}_{n-r} & \boldsymbol{U} \end{pmatrix} \left(\sum_{i=1}^{m} x_i \boldsymbol{M}_i \right) \\ &\quad = \begin{pmatrix} \boldsymbol{v}_{J,T} & \boldsymbol{v}_{J,T}\boldsymbol{U} \end{pmatrix} \left(\sum_{i=1}^{m} x_i \boldsymbol{M}_i \right) \\ &\quad = \left(\sum_{j \in J} \det\left(\boldsymbol{U}_{[J\setminus\{j\},T]}\right) \boldsymbol{e}_j^{n-r} \quad \sum_{s \notin T} \det\left(\boldsymbol{U}_{[J,T\cup\{s\}]}\right) \boldsymbol{e}_s^{r} \right) \left(\sum_{i=1}^{m} x_i \boldsymbol{M}_i \right) \\ &\quad = \sum_{i=1}^{m} \left[\sum_{j \in J} x_i \boldsymbol{M}_{i,[\ell,j]} \det\left(\boldsymbol{U}_{[J\setminus\{j\},T]}\right) + \sum_{s \notin T} x_i \boldsymbol{M}_{i,[\ell,n-r+s]} \det\left(\boldsymbol{U}_{[J,T\cup\{s\}]}\right) \right]_{\ell=1,\ldots,n} \end{aligned}$$

From this last line, we can see that for a product of the system by a vector of minors of degree d, we obtain monomials of two bidegrees : $(1, d)$ and $(1, d+1)$. Their respective coefficient that are going to be in the Macaulay matrix are elements of the matrix $\boldsymbol{M}_i$.

For a row of degree d, corresponding to the multiplication of the system by minors of degree d, we have two adjacent blocks: one of degree $d+2 =$ bidegree

$(1, d+1)$ and one of degree $d+1$ = bidegree $(1, d)$. This is not the case for the top row of degree r: because of the jacobian matrix, it is not possible to have monomials of bidegree $(1, r+1)$.

Table 3.1: General structure of the Macaulay matrix, with degrees of the minors to multiply for rows and total degree for the columns.

	$\deg r+1$	$\deg r$	$\dots$	$\deg d+2$	$\deg d+1$	$\dots$	$\deg 3$	$\deg 2$	$\deg 1$
$\deg r$	$(1,r)$								
$\deg r-1$	$(1,r)$	$(1,r-1)$							
$\vdots$			$\ddots$						
$\deg d$				$(1,d+1)$	$(1,d)$				
$\vdots$						$\ddots$			
$\deg 1$							$(1,2)$	$(1,1)$	
$\deg 0$								$(1,1)$	$(1,0)$

All the blank cells are zero blocks, as there are a limited variables and degree for each equation of the system. On every row, there is only two nonzero-blocks, corresponding to the colored cells. Although those are not full of zero, they are sparse matrices.

We want to define a way to index this matrix, to make some future analysis easier. For columns, we use 3 indices i, K and T', and a column corresponds to a monomial $x_i \det(\boldsymbol{U}_{K,T'})$ with $i = \{1, \dots, m\}$, $K \subset \{1, \dots, n-r\}$ of size $d+1$ such that $1 \leq k_\ell < k_{\ell+1} \leq n-r$ and T' is as defined before. For rows, we use 3 indices as well, ℓ, J and T. A row then corresponds to a product between a minor of $v_{J,T}$ and a polynomial f_ℓ from the system with $\ell \in \{1, \dots, n\}$ and J and T as previously defined. We chose an order for each set of indices, which can be seen in Table 3.2. For columns, we start by picking a set K, then a set T' and the last thing to be fixed is i. We have the same kind of order for the rows, with J, then T and finally ℓ.

For a given d, we call *block* the submatrix composed of the columns indexed by a same K and the rows indexed by a same J. Those blocks are not zero only if $K = J$. Indeed, the monomials that can be found in the product between a minor and a polynomial of the system are necessarily using the same set of indices, that is to say the same choice of columns and rows in the matrix of variables. This is repeated for all $\binom{n-r}{d+1}$ sets J vertically and for all $\binom{n-r}{d+1}$ sets K horizontally. The same thing is repeated for all degrees in the matrix.

For a given degree, we saw that we have 2 adjacent block matrices and that their entries are either 0 or entries of $\boldsymbol{M}_i$ matrices. To summarize, the first block contains $\boldsymbol{M}_{i,[\ell,s]}$ entries and is indexed by a degree d set in rows and columns while the second block contains $\boldsymbol{M}_{i,[\ell,j]}$ entries and is indexed by a degree d for rows but $d-1$ for columns.

Thanks to the order of indices that we chose, we can notice that the first nonzero blocks for the rows of a degree d have a familiar structure. The next subsection explains this resemblance.

Table 3.2: Part of the matrix that highlights the order of the indices

			K_1						
			T'_1			$\cdots$	$T'_{\binom{r}{d+1}}$		
			$i=1$ $x_1\lvert U_{[K_1,T'_1]}\rvert$	$\cdots$ $\cdots$	$i=m$ $x_m\lvert U_{[K_1,T'_1]}\rvert$		$i=1$ $x_1\lvert U_{[K_1,T'_{\binom{r}{d+1}}]}\rvert$	$\cdots$ $\cdots$	$i=m$ $x_m\lvert U_{[K_1,T'_{\binom{r}{d+1}}]}\rvert$
J_1	T_1	$\ell=1$ $\lvert U_{[J_1,T_1]}\rvert f_1$							
		$\vdots$							
		$\ell=n$ $\lvert U_{[J_1,T_1]}\rvert f_n$							
	$\vdots$								
	$T_{\binom{r}{d}}$	$\ell=1$ $\lvert U_{[J_1,T_{\binom{r}{d}}]}\rvert f_1$							
		$\vdots$							
		$\ell=n$ $\lvert U_{[J_1,T_{\binom{r}{d}}]}\rvert f_n$							

3.3.1.2 Relation with a previous modeling

The block represented in Table 3.2 corresponds to one matrix $\boldsymbol{B}$ that we described in 6.2.4. Indeed, we can see that, like $\boldsymbol{B}$, the block is indexed by T for rows and by T' for columns. We recall the expression of an equation of the system multiplied by a vector of minors:

$$\sum_{i=1}^{m}\left[\sum_{s\notin T} x_i \boldsymbol{M}_{i,[\ell,s]} \det\left(\boldsymbol{U}_{[J,T\cup\{s\}]}\right) + \sum_{j\in J} x_i \boldsymbol{M}_{i,[\ell,j]} \det\left(\boldsymbol{U}_{[J\setminus\{j\},T]}\right)\right]_{\ell=1,\ldots,n} \tag{3.10}$$

We notice that the coefficients that we will have in the matrix are $\boldsymbol{M}_{i,[\ell,j]}$ and $\boldsymbol{M}_{i,[\ell,s]}$, and that the latter actually correspond to entries in the $\boldsymbol{B}$, as given by the definition of its nonzero blocks:

$$\boldsymbol{B}_s = \left(\boldsymbol{M}_{\ell,[*,s]}\right)_{\ell=\{1\ldots n\}} \quad \text{if } T'\setminus T = \{s\}$$

We suppose that the sets J and K are sorted the same way, thus $J_i = K_i$. Then, we can rewrite the first block of the Macaulay matrix for a degree d using the $\boldsymbol{B}$ matrices:

Table 3.3: Block of the Macaulay matrix using matrices $\boldsymbol{B}_i$ for a given degree d

	K_1	K_2	$\cdots$	$K_{\binom{n-r}{d+1}}$
J_1	$\boldsymbol{B}_1$	$\boldsymbol{0}$	$\boldsymbol{0}$	$\boldsymbol{0}$
J_2	$\boldsymbol{0}$	$\boldsymbol{B}_2$	$\boldsymbol{0}$	$\boldsymbol{0}$
$\vdots$	$\boldsymbol{0}$	$\boldsymbol{0}$	$\ddots$	$\boldsymbol{0}$
$J_{\binom{n-r}{d+1}}$	$\boldsymbol{0}$	$\boldsymbol{0}$	$\boldsymbol{0}$	$\boldsymbol{B}_{\binom{n-r}{d+1}}$

We know the rank of the matrix $\boldsymbol{B}_i$ with $i \in \left\{1,\ldots,\binom{n-r}{d+1}\right\}$ from Conjecture

3.9 if the number of rows is greater than the number of columns:

$$\text{Rank}\left(\boldsymbol{B}\right) = \binom{r}{d+1}n \tag{3.11}$$

We can then deduce the total rank of the submatrix of Table 3.3:

Lemma 3.11. *Let d be an integer such that $0 < d+1 < \min\{n-r, r\}$. The rank of a diagonal block of the Macaulay matrix made of matrices $\boldsymbol{B}_i$ is:*

$$\text{Rank}\left(\text{diag}\left(\boldsymbol{B}_i\right)_{i=1,\ldots,\binom{n-r}{d+1}}\right) = \binom{n-r}{d+1}\binom{r}{d+1}n \tag{3.12}$$

Knowing this rank will help calculating the rank of the full matrix.

3.3.1.3 Rank of the Macaulay Matrix

We want to compute the rank of the whole Macaulay matrix. We remind that we are working on generic systems and thus the ranks given are generic. We will see in the second part that there exist cases where the ranks are actually lower. We need to begin by finding the rank of the two blocks of matrices in all the rows for a degree d.

We already know that the first block has a rank $\binom{n-r}{d+1}\binom{r}{d+1}n$. We need to compute the rank of the adjacent block, the second on the row. To be able to apply Theorem 6.6 and Conjecture 3.9, we focus on the cases where we have more rows than columns.

Let us first recall the expression that gives the coefficients of this second block:

$$\sum_{i=1}^{m}\left[\sum_{j\in J} x_i \boldsymbol{M}_{i,[\ell,j]}\det\left(\boldsymbol{U}_{[J\setminus\{j\},T]}\right)\right]_{\ell=1,\ldots,n}$$

Here, the blocks are zero when T (indexing rows) and T' (indexing columns) are different or when $K \neq J\setminus\{j\}$. We can notice that for a given K, we have multiple couples (J,j) such that $J\setminus\{j\} = K$. For all J and K, we have :

$$\boldsymbol{R}_{J,K} = \begin{cases} \boldsymbol{R}_j & \text{if } K = J\setminus\{j\} \\ \boldsymbol{0} & \text{else} \end{cases}$$

with $\boldsymbol{R}_j$ a block that we will define right after.

Let us first see an example of this structure:

Example 3.12. Let $n-r = 4$ and $d = 1$. The rows are indexed by sets of size $d+1 = 2$ and the columns by sets of size $d = 1$. The block that we then have is:

We notice that the general structure of this matrix is similar to the transpose of a $\boldsymbol{B}$ matrix in 3.2.2.1.

Table 3.4:

	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$
$\{1,2\}$	$\boldsymbol{R}_2$	$\boldsymbol{R}_1$	$\mathbf{0}$	$\mathbf{0}$
$\{1,3\}$	$\boldsymbol{R}_3$	$\mathbf{0}$	$\boldsymbol{R}_1$	$\mathbf{0}$
$\{2,3\}$	$\mathbf{0}$	$\boldsymbol{R}_3$	$\boldsymbol{R}_2$	$\mathbf{0}$
$\{1,4\}$	$\boldsymbol{R}_4$	$\mathbf{0}$	$\mathbf{0}$	$\boldsymbol{R}_1$
$\{2,4\}$	$\mathbf{0}$	$\boldsymbol{R}_4$	$\mathbf{0}$	$\boldsymbol{R}_2$
$\{3,4\}$	$\mathbf{0}$	$\mathbf{0}$	$\boldsymbol{R}_4$	$\boldsymbol{R}_3$

By checking the expression of the equations and the previous matrix, we can notice that the only constraint on the sets T and T' is that they must be equal for the associated block to be nonzero. Thus, we will index the columns by the T sets as well. It leads, for given J and K, to the matrix R_j represented in Table 3.5.

Table 3.5: Details on the R_j matrix

		T_1			$\dots$	$T_{\binom{r}{d}}$		
		$i=1$	$\dots$	$i=m$		$i=1$	$\dots$	$i=m$
T_1	$\ell=1$	$\boldsymbol{M}_{1,[1,j]}$	$\dots$	$\boldsymbol{M}_{m,[1,j]}$		0	$\dots$	0
	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$	$\ddots$	$\vdots$
	$\ell=n$	$\boldsymbol{M}_{1,[n,j]}$	$\dots$	$\boldsymbol{M}_{m,[n,j]}$		0	$\dots$	0
$\vdots$					$\ddots$			
$T_{\binom{r}{d}}$	$\ell=1$	0	$\dots$	0		$\boldsymbol{M}_{1,[1,j]}$	$\dots$	$\boldsymbol{M}_{m,[1,j]}$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$	$\ddots$	$\vdots$
	$\ell=n$	0	$\dots$	0		$\boldsymbol{M}_{1,[n,j]}$	$\dots$	$\boldsymbol{M}_{m,[n,j]}$

We can easily notice that the same block is repeated along the diagonal, and we can write it with a Kronecker product:

$$\boldsymbol{R}_j = \boldsymbol{I}_{\binom{r}{d}} \otimes (\boldsymbol{M}_{1,[*,j]} \dots \boldsymbol{M}_{m,[*,j]})$$

The rank corresponds to the minimum between the number of rows and the number of columns. We can give a conjecture similar to Conjecture 3.9.

Conjecture 3.13. *Suppose that $m > n$, the rank of $\boldsymbol{R}_j$ is then:*

$$\mathit{Rank}(\boldsymbol{R}_j) = \binom{r}{d} n$$

As the matrices $\boldsymbol{R}_j$ are all different, this would imply that the complete rank is $\binom{n-r}{d}\binom{r}{d}n$.

We have the rank of the two blocks independently, we can wonder if considering them together will reduce the rank. To answer this question, we can check

the equation 3.10 and notice that for a given n, when $j = s \in J \backslash T$, we have two similar columns, one in the first block and one in the second. However, we can see from the structure of the matrix given in Table 3.1 and in the different illustrations given that we can not cancel a column because of the other elements that would appear on the same column as a consequence of the linear combination.

The total rank depends only on the size of the matrix: if the number of rows is less than the number of columns, then the rank corresponds to the number of rows; if the number of columns is less than the number of rows, then the rank corresponds to the number of columns minus 1.

We can consider the complete Macaulay matrix, or we can chose a submatrix that checks the requirements to reduce the number of computations to do. If we chose a submatrix, we introduce two indices d_b and d_e such that the rows chosen are all consecutive rows in $\{d_b, d_b + 1, \ldots, d_e\}$. We keep all the columns that are not all zero on those rows. Thus, for $\boldsymbol{P}$ a submatrix of the Macaulay matrix that goes from degree d_b to d_e, we obtain the expression for the total rank.

Proposition 3.14. *Let d_b and d_e be two integers such that $0 \leq d_b \leq d_e \leq r$ with r the maximal degree possible. Let $\boldsymbol{P}$ be a Macaulay matrix for the degrees d_b to d_e. The rank of this matrix is:*

$$Rank\ (\boldsymbol{P}) = \min\left\{ \sum_{i=d_b}^{d_e} m\binom{n-r}{i+1}\binom{r}{i}, \left(\sum_{i=d_b}^{d_e} n\binom{n-r}{i}\binom{r}{i} \right) - 1 \right\}$$

3.3.2 Complexity Results

We experimented this method on generic bilinear systems in a field of characteristic 2 and an extension degree 4 over $\mathbb{F}_2$, on two different sizes. Later in this thesis, we will also experiment on the DAGS cryptosystem.

Table 3.6: Results for $m = 10$, $n = 10$ and $\nu = n - r$ and comparison between the method of 3.2 and the method introduced in 3.3. d is the highest degree reached during the computations, Matrix Size is the matrix we use for our Gröbner basis computations and Time is the duration of those computations. If it is possible, we took a submatrix of the Macaulay matrix to have the smallest matrix possible.

		Classic Method 3.2		Minors Method 3.3		
r	d	Matrix Size	Time	d	Matrix Size	Time
2	3	1984×2209	0.05s		560×280	0.02s
3	4	26502×24582	1.29s		350×350	0.03s
4	4	31206×43930	3.33s		1860×1850	0.44s
5	5	243815×341515	145s		$- \times -$	$-$s
6	-	Too big	-		$- \times -$	$-$s

In table 3.6, we can see a difference when $r = 5$ and $r = 6$. Indeed, we can not find a submatrix of the Macaulay matrix that has more rows than columns. In those case, we can not linearize the system.

For $r = 5$, the classic method still works in a reasonable time.

Table 3.7: Results for $m = 12$, $n = 12$ and $\nu \leq n - r$ and comparison between different methods.

			Classic Method 3.2			Minors Method 3.3	
r	ν	d	Matrix Size	Time	d	Matrix Size	Time
	8	4	72920×98598	20s		9504×5940	31s
	7	4	53641×72785	12s		5544×3960	12s
	6	4	37986×51850	9s		3024×2520	4.6s
4	5	4	33995×29339	16s		1512×1512	1.5s
	4	4	33793×18349	10.6s	-	$- \times -$	−s
	3	5	26959×24798	8.0s	-	$- \times -$	−s
	2	5	42969×34704	31s	-	$- \times -$	−s
	7	5	982072×1297064	2148s		11088×9504	75s
	6	5	630209×838357	1037s		5544×5544	13s
5	5	5	350320×469744	394s	-	$- \times -$	−s
	4	5	227715×200808	499s	-	$- \times -$	−s
	3	6	205830×205140	1050s	-	$- \times -$	−s

We can notice in table 3.7 that the computations get difficult when ν is equal to or lower than r. This is particularly visible on the last case where $r = 6$. Indeed, all the values possible for ν are smaller than 6 and thus the minors method does not seem to be efficient on those cases.

3.3.3 Relation with SupportMinors

In [12], the authors presented a method named SupportMinors, and the method we presented in the previous sections is actually SupportMinors, considering $b = 1$, as explained in [9]. Here we explain what is the SupportMinors modeling and how it is related to previous modelings.

We first need to define the Plücker coordinates:

Definition 3.15 ([41])**.** *Let $\boldsymbol{A}$ be a matrix of size $n_A \times m_A$ with $n_A < m_A$. The* Plücker coordinates *of $\boldsymbol{A}$ correspond to a vector whose entries are the maximal minors of $\boldsymbol{A}$. These minors are of order n_A and there are $\binom{m_A}{n_A}$ of those coordinates.*

We denote a_T the new variables that replace the minors of $\boldsymbol{A}$. This transformation is done using the injective Plücker map [58].

Definition 3.16. *The* Plücker map *can be defined as:*

$$p : \left\{\mathcal{W} \subset \mathbb{F}_q^n : \dim(\mathcal{W}) = r\right\} \to \mathbb{P}^N(\mathbb{F}_q) \qquad (N = \tbinom{n}{r} - 1)$$

$$\boldsymbol{A} \mapsto (a_T)_{T\subset\{1..n\},\#T=r}.$$

Let $\boldsymbol{M}_x = \sum_{\ell=1}^{n_x} x_\ell M_\ell$. We first define a Minors Modeling:

Definition 3.17 (Minors modeling)**.** *Let $(\boldsymbol{M}_1,\dots,\boldsymbol{M}_{n_x}) \in \mathbb{F}_q^{m\times n_y}$ be a MinRank instance with a target rank r. Then, the MinRank problem can be solved by finding $x_1,\dots,x_{n_x} \in \mathbb{F}_q^{n_x}$ such that*

$$\{|\boldsymbol{M}_x| = 0, \forall J \subset \{1,\dots,m\}, \#J = r+1, \forall T = \{t\}\cup\{n-r+1,\dots,n\} \subset \{1,\dots,n\}\} \tag{Minors}$$

From this Minors modeling, we can define two versions of the Support Minors modeling for MinRank instances, corresponding to two different ideas introduced in [12].

The first idea is that the vector space of the generator matrix $\boldsymbol{M}_x$ is orthogonal to the one with generator matrix $\begin{pmatrix}\boldsymbol{I}_\nu & \boldsymbol{C}^T\end{pmatrix}$. This leads to the first Support Minors modeling:

Definition 3.18 (Support Minors modeling)**.** *Let $(\boldsymbol{M}_1,\dots,\boldsymbol{M}_{n_x}) \in \mathbb{F}_q^{m\times n_y}$ be matrices used in a MinRank instance with a target rank r. The problem can be solved by finding $x_1,\dots,x_{n_x} \in \mathbb{F}_q^{n_x}$ and $\boldsymbol{U} = (u_{i,j})_{1\le i\le r, 1\le j\le n-r} \in \mathbb{F}_q^{n-r}$ such that:*

$$\left\{ \left| \begin{pmatrix} \boldsymbol{r}_i \\ -\boldsymbol{U} \quad \boldsymbol{I}_r \end{pmatrix} \right|_{*,T} = 0, \forall T \subset \{1,\dots,n\}, \#T = r+1, \boldsymbol{r}_i \text{ a row of } \boldsymbol{M}_x \right\} \tag{SM-$\boldsymbol{U}$}$$

We obtain $m\binom{n}{r+1}$ equations that have a bidegree $(1,r)$ respectively in the n variables x and the $r\times(n-r)$ variables c.

We notice that all the entries of $\boldsymbol{U}$ in this modeling appear as maximal minors of $(-\boldsymbol{U} \quad \boldsymbol{I}_r)$. Thus, we can replace the minors of degree r in the variables u by new variables u_T using the Plücker coordinates.

Definition 3.19 (Support Minors modeling - Minors version [12])**.**
Let $(\boldsymbol{M}_1,\dots,\boldsymbol{M}_{n_x}) \in \mathbb{F}_q^{\times}$ be matrices used in a MinRank instance with a target rank r. The problem can be solved by finding $x_1,\dots,x_{n_x} \in \mathbb{F}_q^{n_x}$ and $(u_T)_{T\subset\{1,\dots,n\},\#T=r+1} \subset \mathbb{F}_q^{\binom{n}{r}}$ such that:

$$\left\{ \sum_{t\in T} (\boldsymbol{M}_{\boldsymbol{x}})_{t,i}\, u_{T\setminus\{t\}} = 0, \forall T \in \{1,\dots,n\}, \#T = r+1, \text{ and } i \in \{1,\dots,m\} \right\} \tag{SM}$$

We obtain $m\binom{n}{r+1}$ equations that are bilinear in the n variables x and the $\binom{n}{r}$ minor variables u_T.

The advantages of such a modeling is that each minor variable u_T is replacing $r!$ terms of degree r, reducing the size of the Macaulay matrix and making

the computations easier. It can be considered as a mix between the previous approach in 3.3 and the first Support Minors modeling.

The Support Minors modeling is currently the most efficient one to solve systems from a MinRank instance.

The relation between the different modelings can also be highlighted by the relations between their ideals. The following proposition illustrates that.

Proposition 3.20. *The equations from the* (Minors) *modeling are included in the ideal generated by the equations of the* (KS) *modeling.*

Proof. We have seen before in Lemma 3.3 and Proposition 3.7 that the jacobian matrices for the (KS) modeling have a very particular form. Indeed, we can write $\boldsymbol{M}_x = \begin{pmatrix}\boldsymbol{M}_x^{(1)} & \boldsymbol{M}_x^{(2)}\end{pmatrix}$, with $\boldsymbol{M}_x^{(1)}$ and $\boldsymbol{M}_x^{(2)}$ matrices of respective size $m \times n-r$ and $m \times r$. We can write every matrix $\boldsymbol{M}_\ell$ the same way: $\boldsymbol{M}_\ell = \begin{pmatrix}\boldsymbol{M}_\ell^{(1)} & \boldsymbol{M}_\ell^{(2)}\end{pmatrix}$
We notice that $\boldsymbol{M}_x^{(2)}\boldsymbol{U}$ is the homogeneous part of highest degree of the system and we can write the following Jacobian matrices:

$$\begin{aligned}\mathrm{jac}_{\mathrm{vec}_{row}(\boldsymbol{U})}\left(\mathrm{vec}_{col}\left(\boldsymbol{M}_x^{(2)}\boldsymbol{U}\right)\right) &= \boldsymbol{I}_{n-r} \otimes \boldsymbol{M}_x^{(2)} \\ \mathrm{jac}_{\boldsymbol{x}}\left(\mathrm{vec}_{row}\left(\boldsymbol{M}_x^{(2)}\boldsymbol{U}\right)\right) &= (\boldsymbol{I}_m \otimes \boldsymbol{U})\left(\mathrm{vec}_{row}\left(\boldsymbol{M}_\ell^{(2)}\right)\right)_{\ell=\{1\ldots m\}}\end{aligned}$$

The jacobian matrix in $\boldsymbol{U}$ admits a left kernel containing the following vectors:

$$\boldsymbol{e}_i \otimes \boldsymbol{v}_J(\boldsymbol{M}_x^{(2)}) \text{ for any } J \subset \{1..m\}, \#J = r+1, 1 \leq i \leq n-r$$

where $\boldsymbol{e}_i$ corresponds to the ith row of $\boldsymbol{I}_{n-r}$. Then we can see that the ideal generated by the (KS) equations contains the following equations:

$$\begin{aligned}(\boldsymbol{e}_i \otimes \boldsymbol{v}_J(\boldsymbol{M}_x^{(2)}))\mathrm{vec}_{col}\left(\boldsymbol{M}_x\begin{pmatrix}\boldsymbol{I}_{n-r}\\ \boldsymbol{U}\end{pmatrix}\right) &= (\boldsymbol{e}_i \otimes \boldsymbol{v}_J(\boldsymbol{M}x^{(2)}))\mathrm{vec}_{col}\left(\boldsymbol{M}_x^{(1)}\right) \\ &= \boldsymbol{v}_J(\boldsymbol{M}_x^{(2)})\boldsymbol{M}_x^{(1)}\boldsymbol{e}_i^T \\ &= |\boldsymbol{M}_x|_{J,\{i\}\cup\{n-r+1..n\}}\,.\end{aligned}$$

Those are the (Minors) equations. We can remark that the ideals generated by the (KS) equations and by the (Minors) equations can not be equal, as the (Minors) equations contains only one set of variables. □

Proposition 3.21. *The equations of the* (KS) *modeling are included among the equations of the* (SM-$\boldsymbol{U}$) *modeling, and their ideals are equal.*

Proof. We use similar notation that in the previous proof.

The jacobian matrix in x variables admits a left kernel containing the following vectors:

$$\boldsymbol{e}_\ell \otimes \boldsymbol{v}_J(\boldsymbol{U}) \text{ for any } J \subset \{1..n-r\}, \#J = r+1, 1 \leq \ell \leq m$$

With the same reasoning as before, we can say that the ideal generated by the (KS) equations contains the following equations:

$$\begin{aligned}(\boldsymbol{e}_\ell \otimes \boldsymbol{v}_J(\boldsymbol{U}))\text{vec}_{row}\left(\boldsymbol{M}_x^{(1)}\right) &= \boldsymbol{e}_\ell \boldsymbol{M}_x^{(1)} \boldsymbol{v}_J(\boldsymbol{U})^T \\ &= \boldsymbol{v}_J(\boldsymbol{U})\left(\boldsymbol{M}_{x,[\ell,*]}^{(1)}\right)^T \\ &= \left|\left(\left(\boldsymbol{M}_{x,[\ell,*]}^{(1)}\right)^T \quad \boldsymbol{U}^T\right)\right|_{J,*} \\ &= \left|\begin{pmatrix}\boldsymbol{M}_{x,[\ell,*]}^{(1)} \\ \boldsymbol{U}\end{pmatrix}\right|_{*,J}\end{aligned}$$

We just found exactly the (SM-$\boldsymbol{U}$) equations for $J \subset \{1 \dots n-r\}$, that have a degree r in the variables $u_{i,j}$.

In Theorem 3.8, we try to solve (KS) instances by finding elements in the left kernel of the jacobian matrix in x for some degree $1 < d \leq r-1$. To do so, we consider all combinations of the polynomials with coefficients

$$\boldsymbol{e}_\ell \otimes \boldsymbol{v}_J(\boldsymbol{U}_{T,*})$$

for any $d \in \{1,\dots,r\}$, $J \subset \{1,\dots,n-r\}$, $\#J = d+1$, $T \subset \{1,\dots,r\}$, $\#T = d$ and $\ell = \{1,\dots,m\}$.

We can remark that in Theorem 3.8, we consider the following equations:

$$\begin{aligned}(\boldsymbol{e}_\ell \otimes \boldsymbol{v}_J(\boldsymbol{U}_{T,*}))\text{vec}_{row}\left(\boldsymbol{M}_x\begin{pmatrix}\boldsymbol{I}_{n-r} \\ \boldsymbol{U}\end{pmatrix}\right) &= \boldsymbol{e}_\ell \boldsymbol{M}_x\begin{pmatrix}\boldsymbol{I}_{n-r} \\ \boldsymbol{U}\end{pmatrix}(\boldsymbol{v}_J(\boldsymbol{U}_{T,*}))^T \\ &= \left|\begin{pmatrix}\boldsymbol{r}_\ell \\ -\boldsymbol{U} \quad \boldsymbol{I}_r\end{pmatrix}\right|_{*,T'}\end{aligned}$$

for $T' = J \cup (\{n-r+1..n\} \setminus (T+n-r)) \subset \{1..n\}$ of size $r+1$. The equations have a degree d in the variables $u_{i,j}$.

For $d = 0$ and $T' = \{\ell\} \cup \{n-r+1..n\}$, $\left|\begin{pmatrix}\boldsymbol{r}_\ell \\ -\boldsymbol{U} \quad \boldsymbol{I}_r\end{pmatrix}\right|_{*,T'}$ corresponds to the ℓth equation from the (KS) modeling, and we get all the (SM) equations. □

Remark 3.22. Considering Propositions 3.20 and 3.21, we are able to get a better understanding of the behavior of a generic Gröbner basis algorithm with a graded monomial ordering. As (SM-$\boldsymbol{U}$) contains (KS) directly into the system, computing a Gröbner basis on (SM-$\boldsymbol{U}$) will also compute all equations that would be computed by (KS). On the other hand, when computing a Gröbner basis for (KS), the algorithm will produce all equations from (SM-$\boldsymbol{U}$) by multiplying by monomials in $\boldsymbol{U}$, hence we can expect many syzygies during a Gröbner basis computation on (SM-$\boldsymbol{U}$).

It seems to indicate that the best strategy would be to compute with (SM-$\boldsymbol{U}$), but to look only at multiple of the equations by the x_i's variables, which is the strategy proposed in [12]. Moreover, by considering minors as variables as we previously explained, it removes the hardness of computing with high degree polynomials.

Part II

Attacking Cryptosystems

Chapter 4

Code-Based Cryptography

Contents

4.1 Introduction

Historically, cryptography has been restricted to what is called *secret key cryptography* or *symmetric cryptography*. This kind of cryptography implies that the same secret key is used to encrypt and decrypt a message, and this secret key is shared only to the people that are authorized to read the message.

However, in the 70s, an other family emerged: *public key cryptography*, also called *asymmetric cryptography*. Contrary to the secret key cryptography, the public key cryptography uses a pair of keys: one is a *public key* and can be known by anybody, the other is a *private key* and is only known to one person. This kind of cryptography can be used for multiple purposes, including encryption and signature. In the first case, the public key is used by the sender to encrypt and the private key by the receiver to decrypt the message. In the second case, the person who wants to sign uses its own private key, and any other person can verify the authenticity of the signature with the public key.

Public-key cryptography was introduced by Diffie and Hellman in 1976 [27] but the first scheme was proposed in 1978 by Rivest, Shamir and Adleman [55]. This cryptosystem is called RSA, and its security rests on a problem considered as hard: factorizing large numbers, that is to say finding the prime factors of such a large number. It is one of the main schemes that are used nowadays. Since it was published, multiple other systems were proposed using the same problem, but some were based on other number theory problems such as the discrete logarithm problem.

However, those algorithms are all vulnerable to quantum computers that are currently emerging, particularly to Shor's algorithm [56] that allows to quickly find prime factors of an integer. To respond to this threat, the public key cryptography was updated with the new family of *post-quantum cryptography*. The aim of this cryptography is to provide algorithms that are resistant to quantum computers, while still be usable by traditional ones. To do so, they rely on problems that are considerd hard for quantum computers. In this thesis, we will consider a subfamily of post-quantum cryptography that is using error-correcting codes.

Outline of the chapter

In this chapter, we will present all the basic definitions about codes and their link to cryptography. In section 4.2, we will talk about the coding theory, in particular linear codes, operations on codes and different families of codes that we will encounter later in this thesis. Then, in section 4.3, we will cover the subject of code-based cryptography: its foundations with the McEliece scheme and the basic attacks that can be used against it.

4.2 Coding Theory

In this section, we will introduce definitions, properties and theorems about error-correcting codes, with a focus on linear codes. The main reference for this matter is [48].

Let $\mathbb{F}_q$ be a finite field of q elements.

4.2.1 Linear codes

There exist multiple types of codes, and here we will mostly talk about linear codes.

Definition 4.1. *Let k and n be two integers such that $0 < k \leq n$. A* linear code $\mathscr{C}$ *of length n and dimension k is a linear subspace of $\mathbb{F}^n$ of dimension k. $\mathscr{C}$ is said to be an $[n, k]$-code.*

A vector of $\mathscr{C}$ is called a *codeword*. The ratio $\frac{k}{n}$ is called the *information rate* and the difference $n - k$ is the *redundancy* of the code.

4.2.1.1 Generator and Parity-Check Matrices

A linear code can be defined by two different matrices: its generator matrix or its parity-check matrix.

Definition 4.2. *A* generator matrix $\boldsymbol{G}$ *of a code $\mathscr{C}$ is a matrix whose rows form a basis of the code $\mathscr{C}$. We can write:*

$$\mathscr{C} = \left\{ \boldsymbol{mG} \mid \boldsymbol{m} \in \mathbb{F}_q^k \right\}$$

For a given $\mathscr{C}$, there are multiple generator matrices. However, some matrices form are more interesting than others to work with, such as the *systematic form*. A generator matrix is said to be in systematic form when it is written as:

$$\boldsymbol{G} = (\boldsymbol{I}_k \mid \boldsymbol{A})$$

with $\boldsymbol{A}$ a matrix of size $k \times (n - k)$ over $\mathbb{F}_q$. Such a matrix does not always exist, but when it does, it is unique. In this thesis, we will usually assume that it does exist.

Let us now define parity-check matrices.

Definition 4.3. *A* parity-check matrix $\boldsymbol{H}$ *of a code $\mathscr{C}$ is a matrix of size $(n - k) \times k$ over $\mathbb{F}_q$ of rank $(n - k)$ such that:*

$$\forall \boldsymbol{c} \in \mathscr{C}, \boldsymbol{Hc}^T = \boldsymbol{0}$$

As it is the case for generator matrices, there exist multiple parity-check matrices for a given code.

The generator matrix and the parity-check matrix for a given code $\mathscr{C}$ are linked. If the generator matrix $\boldsymbol{G}$ is written as $\begin{pmatrix} \boldsymbol{I}_k & | & \boldsymbol{P} \end{pmatrix}$ then an associated

parity-check matrix $\boldsymbol{H}$ is written as $\left(-\boldsymbol{P}^T \mid \boldsymbol{I}_{n-k}\right)$ because we can write that $\boldsymbol{G} \cdot \boldsymbol{H}^T = 0$.

Parity-check matrices can also be used to define the syndrome :

Definition 4.4. *Let $\mathscr{C}$ be an $[n, k]$ code, $\boldsymbol{H}$ its parity-check matrix and $\boldsymbol{c}$ a vector of size n. The* syndrome $\boldsymbol{s}$ *of a vector $\boldsymbol{c}$ is :*

$$\boldsymbol{s} = \boldsymbol{c}\boldsymbol{H}^T$$

This resulting vector $\boldsymbol{s}$ is a vector of size $n - k$. When the syndrome is a zero vector, it means that the vector $\boldsymbol{c}$ is a codeword of the code $\mathscr{C}$.

We will encounter the syndrome later in this chapter, when discussing the Syndrome Decoding Problem in 4.35. Another important fact about parity-check matrices is that they are also the generator matrices of other codes, as explained in the following definition.

Definition 4.5. *For $\mathscr{C}$ a linear code over $\mathbb{F}_q$, we can define its* dual *or* orthogonal *code denoted $\mathscr{C}^\perp$, the set of all vectors orthogonal to all codewords of $\mathscr{C}$:*

$$\mathscr{C}^\perp = \left\{\boldsymbol{y} \in \mathbb{F}_q^n \mid \boldsymbol{x}\boldsymbol{y}^T = 0, \forall \boldsymbol{x} \in \mathscr{C}\right\}$$

A parity-check of the code $\mathscr{C}$ is a generator matrix for its dual $\mathscr{C}^\perp$. $\mathscr{C}$ and $\mathscr{C}^\perp$ have the same length n, however $\mathscr{C}^\perp$ has a dimension of $n - k$ with k the dimension of $\mathscr{C}$.

4.2.2 Building new codes from old

4.2.2.1 Puncturing and Shortening

There are different operations that can be used to create codes from already existing ones. We present two of them: puncturing and shortening. Both imply a reduction of the length of the code, but only the shortening usually induces a reduction of the dimension too.

Definition 4.6. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a code and $\mathcal{I} \subseteq \{1, \ldots, n\}$ be a set of coordinates. The puncturing of the code $\mathscr{C}$ on the coordinates of $\mathcal{I}$ is defined by:*

$$\mathrm{Punct}_{\mathcal{I}}(\mathscr{C}) = \left\{(c_i)_{i \in \{1,\ldots,n\} \setminus \mathcal{I}} \mid \boldsymbol{c} \in \mathscr{C}\right\}$$

The resulting code has a length $n - |\mathcal{I}|$ and its dimension k' and minimal distance d' are in the following intervals:

$$k - |\mathcal{I}| \leq k' \leq k \text{ and } d - |\mathcal{I}| \leq d' \leq d$$

In this case, the parameter k' usually tends to be k and d' tends to be $d - |\mathcal{I}|$. Practically, it can be done using the generator matrix $\boldsymbol{G}$ of a code $\mathscr{C}$ by removing the columns indexed by $\mathcal{I}$.

Definition 4.7. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a code and $\mathcal{I} \subseteq \{1, \ldots, n\}$ be a set of coordinates. The shortening of the code $\mathscr{C}$ on the coordinates of $\mathcal{I}$ is defined by:*

$$\mathrm{Short}_{\mathcal{I}}(\mathscr{C}) = \mathrm{Punct}_{\mathcal{I}}(\{\boldsymbol{c} \in \mathscr{C} \mid \forall i \in \mathcal{I},\, c_i = 0\})$$

The resulting code has a length $n - |\mathcal{I}|$. Its dimension k' and minimum distance d' are bounded as follows:

$$k - |\mathcal{I}| \leq k' \leq k \text{ and } d \leq d'$$

In this case, the parameter k' usually tends to be $k - |\mathcal{I}|$. Practically, it can be done by puncturing the dual code $\mathscr{C}^{\perp}$, that is to say removing the columns indexed by $\mathcal{I}$ of the parity-check matrix $\boldsymbol{H}$ of $\mathscr{C}$.

The following theorem highlights a link between puncturing and shortening a code.

Theorem 4.8. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a code and $\mathcal{I} \subseteq \{1, \ldots, n\}$ be a set of coordinates. We can write:*

$$\mathrm{Short}_{\mathcal{I}}\left(\mathscr{C}^{\perp}\right) = \left(\mathrm{Punct}_{\mathcal{I}}(\mathscr{C})\right)^{\perp}$$

4.2.2.2 Trace code

Other than shortening and puncturing, another way to obtain a new code from a given one, is to use a trace map:

Definition 4.9. *Let a be an element in $\mathbb{F}_{q^m}$. The* trace *is a $\mathbb{F}_q$-linear map that we can define as :*

$$\begin{array}{rccc} \mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}: & \mathbb{F}_{q^m} & \longrightarrow & \mathbb{F}_q \\ & a & \longmapsto & \displaystyle\sum_{j=0}^{m-1} a^{q^j} \end{array}$$

We can extend this definition to the vector $\boldsymbol{a} = (a_1, \ldots, a_n)$ by applying it component-wise :

$$\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\boldsymbol{a}) = \left(\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(a_1), \ldots, \mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(a_n)\right)$$

The fact that the trace map results in an element of $\mathbb{F}_q$ makes it really interesting for computations. This particularity will be used later in the attack of the cryptosystem DAGS we present in 5.3. This map, applied on every codeword of a code $\mathscr{C}$, allow us to compute the *trace code*.

Definition 4.10. *Let $\mathscr{C} \subset \mathbb{F}_{q^m}^n$ be a code of length n over $\mathbb{F}_{q^m}$. We call* trace code *of $\mathscr{C}$ over $\mathbb{F}_q$ the code defined by:*

$$\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\mathscr{C}) = \left\{\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(\boldsymbol{c}) \mid \boldsymbol{c} \in \mathscr{C}\right\}$$

Proposition 4.11. *Let a be an element of $\mathbb{F}_q$ and m the field extension. Then, the trace of a is $\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(a) = ma$.*

Remark 4.12. If the finite field has a characteristic 2 and an extension $m = 2$, then the trace of $a \in \mathbb{F}_q$ is $\mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}(a) = 0$.

When the fields involved are clear, it is possible to omit them in the notation and write the trace code $\mathrm{Tr}(\mathscr{C})$.

4.2.2.3 Subfield subcodes

For the following definitions, we consider codes over a finite extension $\mathbb{F}_{q^m}$ of $\mathbb{F}_q$. In such codes, there is a possibility that all components of some codewords lie over the subfield $\mathbb{F}_q$, which leads to the following definition.

Definition 4.13. *Let $\mathscr{C} \subset \mathbb{F}_{q^m}^n$ be a code of length n over $\mathbb{F}_{q^m}$. We call* subfield subcode *of $\mathscr{C}$ the code $\mathscr{C}|_{\mathbb{F}_q}$ made up of all the codewords of $\mathscr{C}$ whose all entries lie in $\mathbb{F}_q$:*

$$\mathscr{C}|_{\mathbb{F}_q} = \mathscr{C} \cap \mathbb{F}_q^n$$

The resulting code has the same length as the starting code. Its dimension k' and minimum distance d' are bounded as follows:

$$k' \geq n - m\,(n-k) \text{ and } d' \geq d$$

Theorem 4.14 (Delsarte Theorem, [26])**.** *Let $\mathscr{C} \subset \mathbb{F}_{q^m}^n$ be a linear code, then:*

$$\left(\mathscr{C} \cap \mathbb{F}_q^n\right)^{\perp} = \mathrm{Tr}_{\mathbb{F}_{q^m}/\mathbb{F}_q}\left(\mathscr{C}^{\perp}\right)$$

4.2.3 GRS and Alternant codes

In this thesis we will see different families of codes. We will begin by the GRS codes, continue with alternant codes and finish with subfamilies of alternant codes. We chose to present those because there exists an efficient decoding algorithm for them.

4.2.3.1 GRS codes

Definition 4.15 ([54])**.** *Let n be a positive integer, $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ a vector whose entries are pairwise distinct, that we call the* support *of the GRS code and $\boldsymbol{y} \in \mathbb{F}_{q^m}^n$ a vector whose entries are not zero, that we call the* multiplier *of the GRS code. The* Generalized Reed-Solomon (GRS) code *with support $\boldsymbol{x}$ and multiplier $\boldsymbol{y}$, of dimension k, is defined as*

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) = \left\{(y_1 f(x_1), \ldots, y_n f(x_n)) \mid f \in \mathbb{F}_q[z]_{<k}\right\}$$

A GRS code can also be defined by its generator matrix:

$$\boldsymbol{G} = \begin{pmatrix} y_1 & \ldots & y_n \\ y_1 x_1 & \ldots & y_n x_n \\ \vdots & \vdots & \vdots \\ y_1 x_1^{k-1} & \ldots & y_n x_n^{k-1} \end{pmatrix}$$

Remark 4.16. When $\boldsymbol{y} = \mathbb{1}$, the code is called a *Reed-Solomon (RS) code* that we denote $\mathbf{RS}_k(\boldsymbol{x})$. Its generator matrix is then:

$$\boldsymbol{G} = \begin{pmatrix} 1 & \ldots & 1 \\ x_1 & \ldots & x_n \\ \vdots & \vdots & \vdots \\ x_1^{k-1} & \ldots & x_n^{k-1} \end{pmatrix}$$

We can define multiple properties of the GRS codes:

Proposition 4.17. *Let n, k be two positive integers such that $k \leq n$. Let $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ be a support and $\boldsymbol{y} \in \mathbb{F}_{q^m}^n$ a multiplier.*

- *The minimum distance between two codewords of $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$ is $d = n - k + 1$.*
- *The dual of a code $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$ is a GRS code too:*

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})^\perp = \mathbf{GRS}_{n-k}\left(\boldsymbol{x}, \boldsymbol{y}^\perp\right)$$

  with $\boldsymbol{y}^\perp = (y_1^\perp, \ldots, y_n^\perp)$ the n-tuple such that for all j in $[\![1, n]\!]$ we have

$$\left(y_j^\perp\right)^{-1} = y_j \prod_{\ell=1, \ell \neq j}^{n} (x_\ell - x_j).$$

- *It exists an efficient decoding algorithm for the code $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$ that can correct up to $\lfloor \frac{n-k}{2} \rfloor$ errors.*

4.2.3.2 Alternant codes

We consider now a subcode of the GRS codes, by applying Definition 4.13 of a subfield subcode.

Definition 4.18 ([37, 38, 26])**.** *Let $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^n$ be respectively a support and a multiplier. The* alternant code *over $\mathbb{F}_q$ corresponds to the subfield subcode of a GRS code, and is defined as:*

$$\mathscr{A}_{r,q}(\boldsymbol{x}, \boldsymbol{y}) = \mathbf{GRS}_r(\boldsymbol{x}, \boldsymbol{y})^\perp \cap \mathbb{F}_q^n$$

The integer r is called the degree *of the alternant code. We can write $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$ when the value of q is defined. Its length is n, and its dimension k verifies the following equation:*

$$k \geq n - mr$$

We can define multiple properties of the alternant codes:

Proposition 4.19. *Let n, r be two positive integers such that $r \leq n$. Let $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ be a support and $\boldsymbol{y} \in \mathbb{F}_{q^m}^n$ a multiplier.*

- *The minimum distance of $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$ is $d \geq r + 1$.*
- *The dual of a code $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$ is:*

$$\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})^\perp = \mathrm{Tr}\left(\mathbf{GRS}_r(\boldsymbol{x}, \boldsymbol{y})\right)$$

- *As for the GRS codes, it exists an efficient decoding algorithm for $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$ that can correct up to $t = \lfloor \frac{r}{2} \rfloor$ errors.*

Proposition 4.20. *Let m,r be two positive integers. Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_{q^m}^n$ be respectively a support and a multiplier. Then,*

$$\mathrm{Short}_{\mathcal{I}}\left(\mathscr{A}_r\left(\boldsymbol{x}, \boldsymbol{y}\right)\right) = \mathscr{A}_r\left(\boldsymbol{x}_{\mathcal{I}}, \boldsymbol{y}_{\mathcal{I}}\right)$$

with $\boldsymbol{x}_{\mathcal{I}}$ (resp. $\boldsymbol{y}_{\mathcal{I}}$) the vector $\boldsymbol{x}$ (resp. $\boldsymbol{y}$) from which we removed all coordinates indexed by $\mathcal{I}$.

Alternant codes are a large family of codes, and there are many subfamilies that can be interesting. We will now present two of those subfamilies: Goppa codes and Srivastava codes.

4.2.3.3 Goppa Codes

We begin by the family of Goppa codes. This family is important as its binary subfamily is used in the McEliece cryptosystem.

Given a polynomial P and a vector $\boldsymbol{x}$, the notation $P\left(\boldsymbol{x}\right)$ corresponds to the application of the polynomial P on each coordinate of $\boldsymbol{x}$.

Definition 4.21 ([34, 18])**.** *Let $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ be a support. Let $\Gamma \in \mathbb{F}_{q^m}\left[z\right]$ be a polynomial such that all evaluations of Γ on the coordinates of $\boldsymbol{x}$ are not zero: $\Gamma\left(x_i\right) \neq 0$ for every $i \in \{1, \ldots, n-1\}$. The* Goppa code *of support $\boldsymbol{x}$ and associated to the* Goppa polynomial Γ *is defined as:*

$$\mathscr{G}_r\left(\boldsymbol{x}, \Gamma\right) = \mathscr{A}_{r,q}\left(\boldsymbol{x}, \Gamma\left(\boldsymbol{x}\right)^{-1}\right)$$

The integer r is the degree of the polynomial Γ. Its length is n, and its dimension k verifies the following equation:

$$k \geq n - mr$$

The parity-check matrix of a Goppa code can be written as:

$$\boldsymbol{H} = \begin{pmatrix} \Gamma\left(\alpha_1\right)^{-1} & \ldots & \Gamma\left(\alpha_n\right)^{-1} \\ \alpha_1\Gamma\left(\alpha_1\right)^{-1} & \ldots & \alpha_n\Gamma\left(\alpha_n\right)^{-1} \\ \vdots & \ldots & \vdots \\ \alpha_1^{r-1}\Gamma\left(\alpha_1\right)^{-1} & \ldots & \alpha_n^{r-1}\Gamma\left(\alpha_n\right)^{-1} \end{pmatrix}$$

Among Goppa Codes, there is the binary Goppa Codes subfamily, which relies on a binary structure to provide interesting changes in the minimum distance or the correction capacity. This is why this family is the one used in the McEliece cryptosystem in the next section.

Definition 4.22 ([18])**.** *A* binary Goppa code *is a Goppa code whose vector $\boldsymbol{x}$ is in $\mathbb{F}_{2^m}^n$ and whose Goppa polynomial Γ is defined over $\mathbb{F}_{2^m}$.*

Proposition 4.23 ([53])**.** *While keeping the notation we used before, we give some properties that have changed when setting $q = 2$:*

- *The minimum distance of* $\mathscr{G}_r(\boldsymbol{x}, \Gamma)$ *is* $d \geq 2r+1$.
- $\mathscr{G}_r(\boldsymbol{x}, \Gamma) = \mathscr{G}_r\left(\boldsymbol{x}, \Gamma^2\right)$.
- *It exists an efficient decoding algorithm for* $\mathscr{G}_r(\boldsymbol{x}, \Gamma)$ *that can correct up to* r *errors.*

4.2.3.4 Srivastava Codes

Definition 4.24 ([17, 36, 35])**.** *Let* s *and* t *be two integers. Let* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ *and* $\boldsymbol{w} = (w_1, \ldots, w_s)$ *be two vectors of* $n+s$ *distinct elements of* $\mathbb{F}_{q^m}$*, and* $\boldsymbol{z} = (z_1, \ldots, z_n)$ *a vector of nonzero elements of* $\mathbb{F}_{q^m}$*. We can define a* Generalized Srivastava (GS) code *thanks to its parity-check matrix of the form:*

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}_1 \\ \boldsymbol{H}_2 \\ \vdots \\ \boldsymbol{H}_s \end{pmatrix}$$

where the block are built as following, for $i \in \{1, \ldots, s\}$*:*

$$\boldsymbol{H}_i = \begin{pmatrix} \frac{z_1}{\alpha_1 - w_i} & \cdots & \frac{z_n}{\alpha_n - w_i} \\ \frac{z_1}{(\alpha_1 - w_i)^2} & \cdots & \frac{z_n}{(\alpha_n - w_i)^2} \\ \vdots & \vdots & \vdots \\ \frac{z_1}{(\alpha_1 - w_i)^t} & \cdots & \frac{z_n}{(\alpha_n - w_i)^t} \end{pmatrix}$$

We say that this code is of order st*, of length* $n \leq q^m - s$ *and of dimension* $k \geq n - mst$*.*

These Generalized Srivastava codes will particularly interest us, as they are the family of codes used in the scheme DAGS presented in the chapter 5.

Proposition 4.25 ([48])**.** *Generalized Srivastava Codes are a subfamily of alternant codes.*

Proof. Let us consider an alternant code $\mathscr{A}_{r,q}(\boldsymbol{\alpha}, \boldsymbol{y})$ with $r = st$. Let $\alpha_1, \ldots, \alpha_n$, $w_1, \ldots, w_s$ be $n+s$ distinct elements of $\mathbb{F}_{q^m}$ and $z_1, \ldots, z_n$ be nonzero elements of $\mathbb{F}_{q^m}$. We define a polynomial g in x:

$$g_{(\ell-1)t+k}(x) = \frac{\prod_{j=1}^{s}(x - w_j)^t}{(x - w_\ell)^k} \quad \text{with } \ell = 1, .., s \text{ and } k = 1, \ldots, t$$

We also define the elements of the vector $\boldsymbol{y}$ as:

$$y_i = \frac{z_i}{\prod_{j=1}^{s}(\alpha_i - w_j)^t} \quad \text{with } i = 1, \ldots, n$$

By multiplying each element of $\boldsymbol{y}$ with the polynomial g, we obtain:

$$y_i g_{(\ell-1)t+k}(\alpha_i) = \frac{z_i}{(\alpha_i - w_\ell)^k}$$

We can notice that this corresponds to the definition of an alternant code (as it is a subfield subcode of a GRS code) as well as it is the expression of an elements form the parity-check matrix for a Generalized Srivastava code. □

We can now define some properties for Generalized Srivastava codes.

Proposition 4.26 ([48])**.** *Let n, s and t be three positive integers. Let $\boldsymbol{\alpha}$, $\boldsymbol{w}$ and $\boldsymbol{z}$ as introduced in 4.24. As GS codes are a subfamily of alternant codes, their properties derive from the alternant ones:*

- *The minimum distance of $\mathrm{GS}_t(\boldsymbol{\alpha}, \boldsymbol{w}, \boldsymbol{z})$ is $d \geq st + 1$.*
- *The dual and the decoding algorithm for GS codes are the same as for the alternant codes.*

Remark 4.27. When $t = 1$ and $z_i = \alpha_i^\mu$ for some μ , the code is said to be a *Srivastava code*. Its parity-check matrix is then:

$$\boldsymbol{H} = \begin{pmatrix} \frac{\alpha_1^\mu}{\alpha_1 - w_1} & \cdots & \frac{\alpha_n^\mu}{\alpha_n - w_1} \\ \frac{\alpha_1^\mu}{\alpha_1 - w_2} & \cdots & \frac{\alpha_n^\mu}{\alpha_n - w_2} \\ \vdots & \vdots & \vdots \\ \frac{\alpha_1^\mu}{\alpha_1 - w_s} & \cdots & \frac{\alpha_n^\mu}{\alpha_n - w_s} \end{pmatrix}$$

4.2.4 Cyclic and Dyadic codes

In this subsection, we will introduce two different families of codes that can be used to introduce structure into a code and its associated matrices. Such families are interesting because they allow to reduce the storage of the keys, which is a crucial matter in cryptography. The first family we will discuss is the one of cyclic codes, the second is the one of dyadic codes.

4.2.4.1 Cyclic codes

For this whole subsection, let $\mathbb{F}$ be a finite field.

Definition 4.28. *Let s be a positive integer. A* cyclic shift map *σ_s can be defined as:*

$$\begin{array}{rccc} \sigma_s : & \mathbb{F}^s & \longrightarrow & \mathbb{F}^s \\ & (x_0, \ldots, x_{s-1}) & \longmapsto & (x_{s-1}, x_0, \ldots, x_{s-2}) \end{array}$$

Definition 4.29. *Let σ_s be a cyclic shift. A linear code $\mathscr{C}$ of length s is said to be* cyclic *if*

$$\forall \boldsymbol{c} \in \mathscr{C}, \sigma_s(\boldsymbol{c}) \in \mathscr{C}$$

Definition 4.30. *Let n be a multiple of s. We define the* quasi-cyclic shift σ *as :*

$$\begin{array}{ccccccccc} \sigma: & & \mathbb{F}^n & & & \longrightarrow & & \mathbb{F}^n & \\ & \left(\boldsymbol{b}_1 \right. & | & \ldots & | & \left. \boldsymbol{b}_{\frac{n}{s}}\right) & \longmapsto & \left(\sigma_s(\boldsymbol{b}_1) \right. & | \quad \ldots \quad | \quad \left. \sigma_s\left(\boldsymbol{b}_{\frac{n}{s}}\right)\right) \end{array}$$

It is actually the application of the cyclic shift σ_s on the blocks $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\frac{n}{s}}$ of length s.

Definition 4.31. *Let σ be a quasi-cyclic shift, and n a multiple of s. A linear code $\mathscr{C}$ of length n is said to be* quasi-cyclic *if*

$$\forall \boldsymbol{c} \in \mathscr{C}, \sigma(\boldsymbol{c}) \in \mathscr{C}$$

Such a code has a generator matrix $\boldsymbol{G}$ which is said to be block-circulant. *It means that the matrix can be split into $s \times s$ circulant blocks as follows :*

$$\boldsymbol{G} = \left(\begin{array}{c|c|c} \ddots & \vdots & \\ \hline \cdots & \boldsymbol{G}_{i,j} & \cdots \\ \hline & \vdots & \ddots \end{array}\right) \text{ with } \boldsymbol{G}_{i,j} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{s-1} \\ g_{s-1} & g_0 & \cdots & g_{s-2} \\ \vdots & & \ddots & \vdots \\ g_1 & \cdots & g_{s-1} & g_0 \end{pmatrix}$$

We can see that the knowledge of the first row of each block is sufficient to get the complete matrix. This is the reason why adding such a structure in error-correcting codes reduces the storage of cryptographic keys by a factor that corresponds to the length of the cyclic shift s. Thus, instead of storing $k \times n$ coefficients for the generator matrix, we only need to store $\frac{k \times n}{s}$ coefficients.

4.2.4.2 Dyadic codes

Another family of codes that have a repetitive structure is dyadic codes. We start by defining dyadic matrices.

Definition 4.32. *A matrix $2^k \times 2^k$ is said to be* dyadic *if we can write it recursively as:*

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B} & \boldsymbol{A} \end{pmatrix}$$

where each block is itself a $2^{k-1} \times 2^{k-1}$ dyadic matrix.

Definition 4.33. *A code is called* quasi-dyadic *when its parity-check matrix is a block matrix whose blocks are dyadic matrices.*

Example 4.34. Let us consider a quasi-dyadic code with blocks of size 4×4. The length of the code is then necessarily a multiple of 4. A parity check matrix for such a code can be written as :

$$\boldsymbol{H} = \left(\begin{array}{c|c|c} \ddots & \vdots & \\ \hline \cdots & \boldsymbol{H}_{i,j} & \cdots \\ \hline & \vdots & \ddots \end{array}\right) \text{ with } \boldsymbol{H}_{i,j} = \left(\begin{array}{cc|cc} h_0 & h_1 & h_2 & h_3 \\ h_1 & h_0 & h_3 & h_2 \\ \hline h_2 & h_3 & h_0 & h_1 \\ h_3 & h_2 & h_1 & h_0 \end{array}\right)$$

The dyadic blocks must be square matrices, but the complete parity-check matrix can be not square, as long as both its number of rows and columns are multiple of the size of the dyadic blocks, 4 here.

4.3 Code-Based Cryptography

Code-based cryptography relies on the problem of decoding a random linear code. It is currently considered as intractable for quantum computers, which makes those very interesting to study. In the same period that RSA was proposed, in 1978, McEliece published a public key encryption scheme based on linear codes [49]. Due to their publication at the same time, they were competing to be used but RSA was preferred because of the smaller size of the keys. McEliece's scheme made a come back when the problem on which RSA is based was found to be an easy problem for quantum computers to solve. This scheme seems to stay safe against quantum computers, this is why it is an important part of the schemes proposed to NIST Call for proposals [51].

4.3.1 McEliece Scheme

The McEliece scheme [49] is made up of three algorithms: `KeyGen` for key generation, `Encrypt` for encryption and `Decrypt` for decryption.

4.3.1.1 Key Generation `KeyGen`(n, k, t)

This algorithm takes as input the parameters $n, k, t \in \mathbb{N}$ and a finite field $\mathbb{F}_q$ in order to generate the pair of public/private keys necessary for the scheme. The steps of the algorithm are the following ones :

1. Choose a family of linear codes over $\mathbb{F}_q$ for which an efficient decoding algorithm that we denote $\mathcal{D}$ exists. The basic version of the McEliece scheme uses the families of binary Goppa codes.
2. From this family, pick a $[n, k]$ code $\mathscr{C}$ which can correct up to t errors. We call $\mathcal{D}_{\mathscr{C}}$ the decoding algorithm associated to the code $\mathscr{C}$.
3. Let $\boldsymbol{G}$ be a random generator matrix of $\mathscr{C}$.

The algorithm can then output the pair of keys : the public key $pk = (\boldsymbol{G}, t)$ and the private key $sk = \mathcal{D}_{\mathscr{C}}$.

4.3.1.2 Encryption $\texttt{Encrypt}(\boldsymbol{m}, pk)$

This algorithm takes as input a message $\boldsymbol{m} \in \mathbb{F}_q^n$ and the public key in order to compute and return the corresponding ciphertext. The steps of the algorithm are the following ones :

1. Pick a random error vector $\boldsymbol{e} \in \mathbb{F}_q^n$ such that $||\boldsymbol{e}|| = t$
2. Compute $\boldsymbol{y} = \boldsymbol{m}\boldsymbol{G}_{\text{pub}} + \boldsymbol{e}$

The vector $\boldsymbol{y}$ is the ciphertext.

4.3.1.3 Decryption $\texttt{Decrypt}(\boldsymbol{y}, sk)$

This algorithm takes as input a ciphertext $\boldsymbol{y}$ and the private key in order to retrieve the message $\boldsymbol{m}$. The ciphertext is of the form $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{c} \in \mathscr{C}$ the ciphertext without any errors and $\boldsymbol{e}$ the error vector such that $||\boldsymbol{e}|| \leq t$. The steps of the algorithm are the following ones :

1. Compute $\boldsymbol{c} = \mathcal{D}_{\mathscr{C}}(\boldsymbol{y})$ thanks to the private key sk.
2. Deduce $\boldsymbol{m}$ thanks to the private key and the knowledge of $\boldsymbol{c} = \boldsymbol{m}\boldsymbol{G}$.

The vector $\boldsymbol{m}$ is the plaintext we wanted to retrieve.

4.3.1.4 Niederreiter Variant

After the publication of McEliece scheme [49], a variation was introduced by Niederreiter [50]. It induces the following changes in the different elements :

- the public key is $(\boldsymbol{H}_{\text{pub}}, t)$
- the private key is also the decoding algorithm $\mathcal{D}_{\mathscr{C}}$
- the ciphertext $\boldsymbol{c} = \boldsymbol{H}_{\text{pub}}\boldsymbol{m}^T$

The decryption uses the fact that the product $\boldsymbol{G}\boldsymbol{H}^T = 0$.

4.3.2 Security of the scheme

We present two types of attacks here, that can be performed against a McEliece-like system : *message recovery attacks* and *key recovery attacks*.

Message Recovery Attacks A *message recovery attack* aims at recovering the plaintext $\boldsymbol{m}$ when the ciphertext $\boldsymbol{y}$ and the public key $(\boldsymbol{G}_{\text{pub}}, t)$ are known. Such an attack must be reiterated for each ciphertext we want the corresponding plaintext of, which can be bothersome.

Key Recovery Attacks A *key recovery attack* aims at finding an efficient decoding algorithm, the one from the private key or an equivalent algorithm, when knowing the generator matrix of the code. Once an algorithm is found, we can decrypt all ciphertexts, which makes this attack very powerful. Moreover, even when the scheme has multiple private keys that can help decoding, finding one is sufficient to have a successful attack.

Syndrome Decoding Problem and ISD In cryptography, we need to have a hard problem to ensure the security of a scheme. Both McEliece and Niederreiter schemes security rely on the hardness of the *Syndrome Decoding Problem* :

Problem 4.35 (Syndrome Decoding Problem)**.** *Let $\mathscr{C}$ be a $[n, k]$ code over $\mathbb{F}_q$ and $\boldsymbol{H}$ its parity check matrix. Let t be an integer and $\boldsymbol{s} \in \mathbb{F}_q^{n-k}$ be a uniformly random vector. The* Syndrome Decoding Problem *consists in finding a vector $\boldsymbol{e} \in \mathbb{F}^n$ with $||\boldsymbol{e}|| \leq t$ such that $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}^T$.*

The Syndrome Decoding Problem is NP-complete, and it is hard for the quantum computers for now. That is why the McEliece and Niederreiter cryptosystems are considered serious candidates for post-quantum cryptography, in particular the Classic McEliece cryptosystem [19] proposed to the NIST Post-Quantum Call for Submissions [51].

Chapter 5

Presentation of DAGS cryptosystem

Contents

5.1 Introduction

In 2017, the NIST launched a Call for Submissions for post-quantum algorithms. The goal was to find one or multiple algorithms to use against the quantum computer threat, and for the chosen ones, to be standardized. The researched algorithms are for encryption as well as for signatures.

More than 80 teams of researchers from all over the world made a proposition, and 69 were kept for the first round of analysis. The submissions were separated in multiple families : lattice-based, code-based, isogeny-based or multivariate were among them.

Since them, multiple rounds of analysis were done, and recently, in July 2022, a first group of algorithms was chosen to be standardized. For encryption, only the lattice-based candidate CRYSTALS-KYBER was selected, and for signatures the lattice-based algorithms CRYSTALS-Dilithium and Falcon, as well as the hash-based algorithm SPHINCS+ were chosen. No code-based candidate has been already picked, but some of them are still under analysis for a 4th round : Classic McEliece, BIKE and HQC. The isogeny-based algorithm SIKE was also selected for this 4th round.

Among the algorithms that were eliminated in previous rounds, there were some code-based algorithms, and particularly an algorithm named DAGS. It is based on McEliece and uses Generalized Srivastava codes, we will introduce it here. We will also explain the first attack that was made against it [16].

Outline of the chapter

In this chapter, we begin by introducing the properties and parameters of the DAGS cryptosystem in section 5.2. Then, in section 5.3, we will present the first attack that was effective on it.

5.2 Presentation of the DAGS cryptosystem

DAGS is a variant of the McEliece scheme presented in subsection 4.3.1, but the authors incorporated a randomization into their system, following the model of [52]. This choice is made to improve the efficiency of the scheme by using a shorter starting vector in the encapsulation, and to get tighter security bounds. All the elements presented in this section are extracted from the specifications sent to NIST [5, 6] or the article [7].

5.2.1 Construction

The cryptosystem DAGS can be separated into three distinct algorithms : one is for key generation, a second is for the key encapsulation and a third is for the key decapsulation. As the cryptosystem uses finite fields and alternant codes, we need the following parameters :

- q is the size of the finite field
- m is the extension degree
- n is the length of the QD-alternant code
- k is the dimension of this code
- k' is an arbitrary small value which depends on the base field such that $k = k' + k''$
- $s = 2^\gamma$ is the number of elements of the permutation group (γ being the number of generators)
- t is the degree of the alternant code

5.2.1.1 DAGS Key Generation

The first algorithm generates the public and private keys. Before developing the process of key generation, we need to introduce the co-trace function:

1. Generate the vector $\boldsymbol{h} = (h_0, \ldots, h_{n-1})$ of elements of $\mathbb{F}_{q^m}$:
   (a) Choose random distinct nonzero h_0 and h_{2^ℓ} for $\ell = 0, \ldots, \lfloor \log q^m \rfloor$
   (b) Generate the missing elements of $\boldsymbol{h}$ using the following formula :
   $$\frac{1}{h_{i \oplus j}} = \frac{1}{h_i} + \frac{1}{h_j} + \frac{1}{h_0} \tag{5.1}$$
   (c) Return a selection of blocks of dimension s up to length n
2. Build the corresponding Cauchy support :
   (a) Choose a random offset $\omega \xleftarrow{\$} \mathbb{F}_{q^m}$

(b) Create two vectors $\boldsymbol{u} = (u_0, \ldots, u_{s-1})$ and $\boldsymbol{v} = (v_0, \ldots, v_{n-1})$ such that :

$$u_i = \frac{1}{h_i} + \omega \qquad \text{for } i = 0, \ldots, s-1 \tag{5.2}$$

$$v_j = \frac{1}{h_j} + \frac{1}{h_0} + \omega \qquad \text{for } j = 0, \ldots, n-1 \tag{5.3}$$

3. Create the Cauchy matrix $\hat{H}_1 = C(\boldsymbol{u}, \boldsymbol{v})$ with components $C_{ij} = \frac{1}{u_i - v_j}$
4. Build blocks $\hat{h}_i$ for $i = 2, \ldots, t$ by raising elements of $\hat{H}_1$ to the power i
5. Form the matrix $\hat{H}$ by superimposing the blocks in ascending order.
6. Choose random elements $z_i \xleftarrow{\$} \mathbb{F}_{q^m}$ of a vector $\boldsymbol{z}$ such that $z_{is+j} = z_{is}$ for $i = 0, \ldots, n_0 - 1$ and $j = 0, \ldots, s-1$
7. Form the matrix $H = \hat{H} \cdot \mathrm{diag}(\boldsymbol{z})$
8. Transform H into alternant form to obtain the matrix H' (see [48], chap. 12), that is to say transform H to get a matrix whose each coefficient is of the form $H'_{ij} = f_j(\alpha_i)$, with f_j functions from a set $\mathbb{K}$ to $\mathbb{F}_{q^m}$ and α_i elements of $\mathbb{K}$.
9. Project H onto $\mathbb{F}_q$ using the co-trace function to obtain a matrix H_{base}
10. Put H_{base} into systematic form $\begin{pmatrix} M & I_{n-k} \end{pmatrix}$

The *public key* is the generator matrix $G = \begin{pmatrix} I_k & M^T \end{pmatrix}$, while the *private key* is the alternant matrix H' defined at step 8.

5.2.1.2 DAGS Encapsulation

The second algorithm is for encaspulation. It uses three hash functions :

- $\mathcal{G} : \mathbb{F}_q^{k'} \to \mathbb{F}_q^k$, which helps with generating randomness for the scheme
- $\mathcal{H} : \mathbb{F}_q^{k'} \to \mathbb{F}_q^{k'}$, which provides *plaintext confirmation* as explained in [42].
- $\mathcal{K} : \{0,1\}* \to \{0,1\}^\ell$, which gives the shared symmetric key of length ℓ

Now that have seen those necessary hash functions, here are the steps of the encapsulation algorithm :

1. Pick a vector $\boldsymbol{m} \xleftarrow{\$} \mathbb{F}_q^{k'}$
2. Compute $\boldsymbol{r} = \mathcal{G}(\boldsymbol{m})$ and $\boldsymbol{d} = \mathcal{H}(\boldsymbol{m})$
3. Split $\boldsymbol{r}$ into two parts as $(\boldsymbol{\rho} \,||\, \boldsymbol{\sigma})$ then set $\boldsymbol{\mu} = (\boldsymbol{\rho} \,||\, \boldsymbol{m})$
4. Generate the error vector $\boldsymbol{e}$ of length n and weight w from the vector $\boldsymbol{\sigma}$

5. Compute $\boldsymbol{c} = \boldsymbol{\mu}G + \boldsymbol{e}$
6. Compute $\boldsymbol{k} = \mathcal{K}(\boldsymbol{m})$

The ciphertext to be sent is the pair $(\boldsymbol{c}, \boldsymbol{d})$ while the encapsulated key is the vector $\boldsymbol{k}$.

5.2.1.3 DAGS Decapsulation

The last algorithm is for decapsulation and uses the same hash functions described in the encapsulation. It uses the alternant decoding algorithm from [48] on the noisy codeword received in the ciphertext. It requires the parity-check matrix to be in alternant form, as specified during the computation of the keys.

1. Decode $\boldsymbol{c}$ using the private key H' to obtain codeword $\boldsymbol{\mu}'G$ and error $\boldsymbol{e}'$
2. Return $\perp$ if the decoding fails or if the weight of $\boldsymbol{e}'$ is not w
3. Recover $\boldsymbol{\mu}'$ and split it into two parts $(\boldsymbol{\rho} \,||\, \boldsymbol{m}')$
4. Compute $\boldsymbol{r}' = \mathcal{G}(\boldsymbol{m}')$ and $\boldsymbol{d}' = \mathcal{H}(\boldsymbol{m}')$
5. Split $\boldsymbol{r}'$ as $(\boldsymbol{\rho}'' \,||\, \boldsymbol{\sigma}')$
6. Generate the error vector $\boldsymbol{e}''$ of length n and weight w from the vector $\boldsymbol{\sigma}'$
7. If $\boldsymbol{e}' \neq \boldsymbol{e}''$ or $\boldsymbol{\rho}' \neq \boldsymbol{\rho}''$ or $\boldsymbol{d} \neq \boldsymbol{d}'$, return $\perp$
8. Else, retrieve the decapsulated key $\boldsymbol{k} = \mathcal{K}(\boldsymbol{m}')$

5.2.2 Original Sets of Parameters

The first sets of parameters were given in the DAGS specifications [5, 6]. They are shown in Table 5.1.

Table 5.1: DAGS original sets of parameters

	q	m	n	k	γ	t	NIST Security Level
DAGS_1	2^5	2	832	416	4	13	Level 1 (128 bits AES)
DAGS_3	2^6	2	1216	512	5	11	Level 3 (192 bits AES)
DAGS_5	2^6	2	2112	704	6	11	Level 5 (256 bits AES)

The authors provided 3 sets of parameters corresponding to the three security levels asked by the NIST. They called them DAGS_1, DAGS_3 and DAGS_5, and they respectively correspond to the same security as a 128, 192 and 256 bits AES exhaustive search. We can notice that the characteristic is 2 for all the sets of parameters, which means that we will not have to keep the negative signs along computations.

5.3 First attack on DAGS

In 2018, a first attack was designed against the cryptosystem DAGS and presented in [16], which is the main reference for the contents of this section. It is a key-recovery attack that is available in two versions : one combinatorial, one algebraic. Both versions rely on finding a subcode of the public code using an operation called *conductor*. They differ by the method used : the first makes use of partial brute force while the second solves a polynomial bilinear system.

In this section we begin by giving some definitions needed to understand the attack. We then explain the core principle of the attack and the complexity of the two versions. We end by presenting the changes made to the parameters following this attack.

5.3.1 Definitions

We begin by giving some definitions that are needed to do the attack on DAGS.

5.3.1.1 Codes

To proceed with the attack, we have to create some codes from the public parts of the scheme. We introduce here some products as well as codes that uses them.

Definition 5.1. *The* component-wise product *of two vectors* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *in* $\mathbb{F}_q^n$ *is defined as*

$$\boldsymbol{a} \star \boldsymbol{b} = (a_1 b_1, \ldots, a_n b_n)$$

In the same spirit, we can define a component-wise power *such that, for any positive integer* t*, we have*

$$\boldsymbol{a}^{\star t} = \underbrace{\boldsymbol{a} \star \cdots \star \boldsymbol{a}}_{t \text{ times}}$$

For $\boldsymbol{a} \in \mathbb{F}_{q^2}^n$, we recall the trace $\mathrm{Tr}(\boldsymbol{a})$ and the norm $\mathrm{N}(\boldsymbol{a})$ the vectors that result from the application of the trace and the norm maps component by component:

$$\mathrm{Tr}(\boldsymbol{a}) = (a_1 + a_1^q, \ldots, a_n + a_n^q)$$
$$\mathrm{N}(\boldsymbol{a}) = \left(a_1^{q+1}, \ldots, a_n^{q+1}\right)$$

Definition 5.2. *The* Schur product *of two codes* $\mathscr{A}$ *and* $\mathscr{B} \subseteq \mathbb{F}_q^n$ *corresponds to the code generated by all the component-wise products of one codeword from* $\mathscr{A}$ *and one codeword of* $\mathscr{B}$*:*

$$\mathscr{A} \star \mathscr{B} = \langle \boldsymbol{a} \star \boldsymbol{b} \mid \boldsymbol{a} \in \mathscr{A}, \boldsymbol{b} \in \mathscr{B} \rangle_{\mathbb{F}_q}$$

We can define as well the square code *of a code* $\mathscr{A}$ *by:*

$$\mathscr{A}^{\star 2} = \mathscr{A} \star \mathscr{A}$$

This Schur product has a different behavior when applied to GRS and some alternant codes than when it is applied to random linear codes. It thus provides a way to distinguish GRS codes among random ones. We can give the following result:

Theorem 5.3 (Proposition 6 from [16])**.** *Let $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ and $\boldsymbol{y}, \boldsymbol{y}' \in \mathbb{F}_{q^m}^n$ be respectively a support and two multipliers. Let k, k' be two positive integers. We can write:*

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \star \mathbf{GRS}_{k'}(\boldsymbol{x}, \boldsymbol{y}') = \mathbf{GRS}_{k+k'-1}(\boldsymbol{x}, \boldsymbol{y} \star \boldsymbol{y}')$$

with $k + k' - 1 < n$.

5.3.1.2 Conductors

Definition 5.4 ([16])**.** *Let $\mathscr{C}$ and $\mathscr{D}$ be two codes of length n over $\mathbb{F}_q$. The* conductor *of $\mathscr{D}$ into $\mathscr{C}$ is defined as the largest code $\mathscr{Z} \subseteq \mathbb{F}_q^n$ such that $\mathscr{D} \star \mathscr{Z} \subseteq \mathscr{C}$:*

$$\mathbf{Cond}(\mathscr{D}, \mathscr{C}) = \left\{ \boldsymbol{u} \in \mathbb{F}_q^n \mid \boldsymbol{u} \star \mathscr{D} \subseteq \mathscr{C} \right\}$$

Proposition 5.5. *Let $\mathscr{D}, \mathscr{C} \subseteq \mathbb{F}_q^n$ two codes, then*

$$\mathbf{Cond}(\mathscr{D}, \mathscr{C}) = \left(\mathscr{D} \star \mathscr{C}^{\perp}\right)^{\perp}$$

When computing the conductor of a GRS code into another, we obtain a new code, which is much simpler. The following proposition explains that.

Proposition 5.6. *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_{q^m}^n$ be respectively a support and a multiplier. Let $k \leq k'$ be two integers less than n. Then, we can write:*

$$\mathbf{Cond}(\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}), \mathbf{GRS}_{k'}(\boldsymbol{x}, \boldsymbol{y})) = \mathbf{RS}_{k'-k+1}(\boldsymbol{x})$$

This proposition highlights the main advantage of using a conductor: the code obtained uses $\boldsymbol{x}$ but not $\boldsymbol{y}$, which is canceled. This resulting code is a Reed-Solomon code, which depends only on a support vector.

5.3.1.3 Norm-Trace Code

For the following statements, we consider an element $\alpha \in \mathbb{F}_{q^2}$, $\alpha \neq 1$ such that $\mathrm{Tr}(\alpha) = 1$. The couple $(1, \alpha)$ forms a $\mathbb{F}_q$-basis of $\mathbb{F}_{q^2}$.

Definition 5.7. *Let the vector $\boldsymbol{x} \in \mathbb{F}_{q^2}^n$ be a support. The* norm-trace code *$\mathscr{NT} \subseteq \mathbb{F}_q^n$ can be defined as:*

$$\mathscr{NT}(\boldsymbol{x}) = \langle \mathbb{1}, \mathrm{Tr}(\boldsymbol{x}), \mathrm{Tr}(\alpha \boldsymbol{x}), \mathrm{N}(\boldsymbol{x}) \rangle_{\mathbb{F}_q}$$

The dimension of this code is 4.

5.3.1.4 Invariant subcode

For the explanation of the attack, we need to define two new constructions of codes: the folded code and the invariant code.

Definition 5.8. *Let n, s be two positive integers such that s divides n. Let σ be a quasi-cyclic shift of length n as defined in definition 4.30 and σ_s its associated cyclic shift of length s. The* folded map *on $\mathbb{F}$ is:*

$$\begin{array}{rccc} \varphi_s : & \mathbb{F}^n & \longrightarrow & \mathbb{F}^n \\ & (x_1, \ldots, x_n) & \longmapsto & \sum_{i=0}^{s-1} \sigma_s^i (x_1, x_{s+1}, \ldots, x_{n-s+1}) \end{array}$$

Definition 5.9. *Let $\mathscr{C} \subseteq \mathbb{F}^n$ be a quasi-cyclic code with a quasi-cyclic shift of size s. Then, the* folded code *is $\varphi_s(\mathscr{C}) \subseteq \mathbb{F}^{\frac{n}{s}}$.*

Definition 5.10. *Let $\mathscr{C} \subseteq \mathbb{F}^n$ be a quasi-cyclic code with a quasi-cyclic shift σ. Then, the* invariant code *is defined by:*

$$\mathscr{C}^\sigma = \{c \in \mathscr{C} | \sigma(c) = c\}$$

This invariant code has repeated entries, which may induce some redundancy into the systems we will work on. Such redundancy does not change the result we obtain, but it surely slows down the computations and reduces their efficiency.

So we use a variant of this code, that we will call the *punctured invariant code* and that we will write:

$$\overline{\mathscr{C}}^\sigma = \mathrm{Punct}_{\mathcal{I}_s}(\mathscr{C}^\sigma)$$

where $\mathcal{I}_s = \{1, \ldots, n\} \setminus \{1, s+1, \ldots, n-s+1\}$.

Usually, the folded code and the invariant code are not equal, but we have the following lemma from [15]:

Lemma 5.11. *Let s be a positive integer and $\mathscr{C}$ be a quasi-cyclic code with a quasi-cyclic shift of length s. Then, we have:*

$$\varphi_s(\mathscr{C}) \subseteq \overline{\mathscr{C}}^\sigma$$

If s is not a multiple of the characteristic of the field, then $\varphi_s(\mathscr{C}) = \mathscr{C}^\sigma$

Remark 5.12. Given $\mathscr{C}$ a linear code over $\mathbb{F}_{q^m}$ stable under a permutation σ, the invariant and the subfield subcode operations can commute:

$$\left(\mathscr{C} \cap \mathbb{F}_q^n\right)^\sigma = \left\{\boldsymbol{c} \in \mathscr{C} \mid \boldsymbol{c} \in \mathbb{F}_q^n \text{ and } \sigma(\boldsymbol{c}) = \boldsymbol{c}\right\} = \mathscr{C}^\sigma \cap \mathbb{F}_q^n$$

5.3.1.5 The subcode $\mathscr{D}$

To do the attack, we need to introduce a subcode $\mathscr{D}$ of $\mathscr{C}_{\mathrm{pub}}$ that will allow us to compute the norm-trace code, as we will see in the following subsection. This subcode is unknown by the attacker.

Definition 5.13. *Let be $\mathcal{G}$ a permutation group such that $|\mathcal{G}| \leq q$. (This is true for all the cases we treat in this thesis.) We can define the code $\mathscr{D}$ as an invariant subcode of an alternant code:*

$$\mathscr{D} = \mathscr{A}_{r+q}(\boldsymbol{x}, \boldsymbol{y})^{\mathcal{G}}$$

Theorem 5.14. *Under the heuristic presented in [16], we have the following result:*

$$\mathbf{Cond}(\mathscr{D}, \mathscr{C}_{\text{pub}}) = \mathscr{NT}(\boldsymbol{x})$$

Moreover, the code $\mathscr{D}$ has a codimension $\leq \frac{2q}{|\mathcal{G}|}$ in $(\mathscr{C}_{\text{pub}})^{\mathcal{G}}$.

The result of this theorem is verified most of the time when doing experiments.

5.3.2 Principle of the attack

The general principle of this attack is to find the code $\mathscr{D}$ so that we can compute the norm-trace code $\mathscr{NT}$ and then retrieve the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of the original code.

The attack can be divided in three major steps:

1. Compute the invariant subcode $(\mathscr{C}_{\text{pub}})^{\mathcal{G}}$ of the public code $\mathscr{C}_{\text{pub}}$. The public code, by definition, is known so we just have to apply the definition 5.10 of an invariant code on it.

2. Find the unknown subcode $\mathscr{D}$ of $(\mathscr{C}_{\text{pub}})^{\mathcal{G}}$ of codimension $\frac{2q}{|\mathcal{G}|}$ such that

$$\mathbf{Cond}(\mathscr{D}, \mathscr{C}_{\text{pub}}) = \mathscr{NT}(\boldsymbol{x}) \tag{5.4}$$

   This step is the most difficult one, and two ways of achieving it were presented:

   - The first approach aims at finding $\mathscr{D}$ using a brute force search on subcodes of codimension $\frac{2q}{|\mathcal{G}|}$ of $(\mathscr{C}_{\text{pub}})^{\mathcal{G}}$ (Details in 5.3.2.1)
   - The second approach consists in solving a bilinear system to find $\mathscr{D}$ and $\mathscr{NT}(\boldsymbol{x})$ (Details in 5.3.2.2)

   From the code $\mathscr{D}$, we can compute $\mathscr{NT}(\boldsymbol{x})$.

3. Recover $\boldsymbol{x}$ from $\mathscr{NT}(\boldsymbol{x})$ and then $\boldsymbol{y}$ from $\boldsymbol{x}$ (Details in 5.3.2.3)

5.3.2.1 Combinatorial version: brute force search

The first way of finding the subcode $\mathscr{D} \subseteq (\mathscr{C}_{\text{pub}})^{\mathcal{G}}$ is to conduct a brute force search. We enumerate all the subspaces $\mathscr{X}$ of codimension $\frac{2q}{|\mathcal{G}|}$ until we find a $\mathbf{Cond}(\mathscr{X}, \mathscr{C}_{\text{pub}})$ whose dimension is 4. However, the number of such subspaces is unreasonably large to be practically computed.There are two strategies we can use to reduce the cost of this search.

The first one is to use the fact that the public code has a rate less than $\frac{1}{2}$. Hence, there is a high probability that the conductor $\mathbf{Cond}\left(\mathscr{D}_0, \mathscr{C}_{\text{pub}}\right)$, with $\mathscr{D}_0$ a random subcode of $\mathscr{D}$ of dimension 2, equals $\mathscr{NT}\left(\boldsymbol{x}\right)$. It results in an average number of $O\left(q^{\frac{4q}{|\mathcal{G}|}}\right)$ computations to find $\mathscr{NT}\left(\boldsymbol{x}\right)$.

The second one consists in shortening the public code $\mathscr{C}_{\text{pub}}$ on a set of $a \geq 1$ positions corresponding to a_0 blocks of size $|\mathcal{G}|$. This shortened code conveniently remains quasi-dyadic. The integer a is chosen so that the shortening of $\mathscr{D}$ has a dimension 2. We then enumerate the possible subspaces until we find one that has a conductor of dimension 4, that equals $\mathscr{NT}\left(\boldsymbol{x}\right)$ with a high probability. It results in an average number of $O\left(q^{\frac{4q}{|\mathcal{G}|}}\right)$ computations to find $\mathscr{NT}\left(\boldsymbol{x}\right)$, just as the strategy previously presented.

Both strategies allow a reduction of the number of computations needed to find the conductor $\mathscr{NT}\left(\boldsymbol{x}\right)$ that is enough to make this attack doable.

Thus, for each subspace $\mathscr{X}$ that we enumerate, we have to do an average number of $O\left(q^{\frac{4q}{|\mathcal{G}|}}\right)$ computations. Finally, we need to consider the cost of computing the conductor for each $\mathscr{X}$ we have to enumerate, which is at most $2n^3$ computations in $\mathbb{F}_q$. The final complexity is then :

$$O\left(n^3 q^{\frac{4q}{|\mathcal{G}|}}\right) \text{ operations in } \mathbb{F}_q \tag{5.5}$$

These complexity computations are explained in greater details in [16].

5.3.2.2 Algebraic version: polynomial system solving

Another way of finding the subcode $\mathscr{D}$ is to solve a specific bilinear system. To determine this system, we use the facts that $\mathrm{Tr}\left(\boldsymbol{x}\right) \in \mathbf{Cond}\left(\mathscr{D}, \mathscr{C}_{\text{pub}}\right)$ and $\mathbf{Cond}\left(\mathscr{D}, \mathscr{C}_{\text{pub}}\right) = \left(\mathscr{D} \star \mathscr{C}_{\text{pub}}^{\perp}\right)^{\perp}$. From these two formulas, we can write:

$$\boldsymbol{G}_{\mathscr{D}\star\mathscr{C}_{\text{pub}}^{\perp}} \cdot \mathrm{Tr}\left(\boldsymbol{x}\right)^T = 0 \tag{5.6}$$

where $\boldsymbol{G}_{\mathscr{D}\star\mathscr{C}_{\text{pub}}^{\perp}}$ is the generator matrix of the code $\mathscr{D} \star \mathscr{C}_{\text{pub}}^{\perp}$. As this equality stands true when replacing $\mathrm{Tr}\left(\boldsymbol{x}\right)$ by $\mathrm{Tr}\left(\beta\boldsymbol{x}\right)$ for any $\beta \in \mathbb{F}_{q^2}$, we also have:

$$\boldsymbol{G}_{\mathscr{D}\star\mathscr{C}_{\text{pub}}^{\perp}} \cdot \boldsymbol{x}^T = 0 \tag{5.7}$$

This gives us the system we need, although we may need to alter it for analysis. We have two unknowns: the code $\mathscr{D}$ and the vector $\boldsymbol{x}$ that will provide us two sets of variables. We denote $c = \frac{2q}{|\mathcal{G}|}$ the codimension of $\mathscr{D}$ in $\left(\mathscr{C}_{\text{pub}}\right)^{\mathcal{G}}$ and $\boldsymbol{G}_{\text{inv}}$ the $k_0 \times n_0$ generator matrix of $\left(\mathscr{C}_{\text{pub}}\right)^{\mathcal{G}}$.

For the code $\mathscr{D}$, we introduce $\left(k_0 - c\right)c$ formal variables in a matrix:

$$\boldsymbol{U} = \begin{pmatrix} U_{1,1} & \cdots & U_{1,c} \\ \vdots & & \vdots \\ U_{k_0-c,1} & \cdots & U_{k_0-c,c} \end{pmatrix}$$

Then, it is probable that the code $\mathscr{D}$ has a generator matrix of the form $\left(\boldsymbol{I}_{k_0-c} \quad | \quad \boldsymbol{U}\right) \cdot \boldsymbol{G}_{\text{inv}}$. If it is not the case, which is rare, we can simply choose another generator matrix for $(\mathscr{C}_{\text{pub}})^{\mathcal{G}}$.

To stay true to the system given above, we search for a generator matrix for $\mathscr{D} \star \mathscr{C}_{\text{pub}}^{\perp}$. We can get one by constructing a matrix whose rows corresponds to all the Schur products of one row of the generator matrix of $\mathscr{D}$ by one row of the parity-check matrix $\boldsymbol{H}_{\text{pub}}$ of $\mathscr{C}_{\text{pub}}$. This matrix is a generator matrix of $\mathscr{D} \star \mathscr{C}_{\text{pub}}^{\perp}$ for a specialization of the variables $U_{i,j}$.

We need to define a product that we will call *star product*, that we will use right after in the writing of the system.

Definition 5.15. *The star product of two matrices $\boldsymbol{A}$ of size $n_A \times m_A$ and $\boldsymbol{B}$ of size $n_B \times m_B$ corresponds to the component-wise product of every combination of one row from each matrix. It is defined as:*

$$(\boldsymbol{A} \star \boldsymbol{B})_{i,j} = a_{(i-1) \text{ div } n_B+1,j} b_{(i-1) \bmod n_B+1,j}$$

It works only if $m_A = m_B$ and the resulting matrix is $n_A n_B \times m_A$. For this specific product, an order has to be chosen for the rows that are taken. The choice here is taking one row on the first matrix, and multiplying it by every row of the second matrix.

The second set of variables corresponds to the entries of the vector $\boldsymbol{x}$ that we will denote $X_1, \ldots, X_n$. Our final system can be written:

$$\left(\left(\left(\boldsymbol{I}_{k_0-c} \quad | \quad \boldsymbol{U}\right) \cdot \boldsymbol{G}_{\text{inv}}\right) \star \boldsymbol{H}_{\text{pub}}\right) \cdot \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = 0 \tag{5.8}$$

The intuitive idea to make the solving of the system easier is to maximize the number of equations while minimizing the number of variables. We will see later that there are other details we need to consider to be the most efficient possible.

For the $(k_0 - c)\, c$ variables in $\boldsymbol{U}$, the only thing we can do is keeping a limited amount of rows in $\boldsymbol{U}$, the minimum, 2, if it is possible. We then have *Number of rows* $\times c$ variables U.

For the n variables X, we can reduce their number by applying three tricks:

1. We use the fact that the code is quasi-dyadic and take advantage of the effects of the additive groupe $\mathcal{G}$. We introduce the variables $A_1, \ldots, A_\gamma$ the generators of $\mathcal{G}$, which allow us to substitute $(X_1, \ldots, X_n)$ by:

$$\begin{aligned} \boldsymbol{V} &= (T_1, T_1 + A_1, \ldots, T_1 + A_1 + \cdots + A_\gamma, T_2, T_2 + A_1, \ldots) \\ &= (T_1, \ldots, T_{n_0}) \otimes \mathbb{1}_{2^\gamma} + \mathbb{1}_{n_0} \otimes \left(0, A_1, A_2, A_1 + A_2, \ldots, \sum_{i=1}^{\gamma} A_i\right) \end{aligned}$$

   for some variables $T_1, \ldots, T_{n_0}$.

2. We are able to specialize two variables thanks to the 2-transitive action of the affine group on $\mathbb{F}_{q^2}$. Then, one can replace variable T_{n_0} by 0 and A_1 by 1.

3. As in the second strategy described in 5.3.2.1, it is possible to shorten the codes in order to reduce the number of variables. We shorten the code $\mathscr{D}$, which also implies that we puncture the matrix $\boldsymbol{H}_{\text{pub}}$ and the vector $\boldsymbol{V}$ accordingly. These changes reduce the length of the support of the code, and the number of the variables T_i accordingly. We will see later on that although it intuitively seems the best idea to reduce the variables to their minimum (which corresponds to keeping only 2 rows of the matrix $\boldsymbol{U}$), this does not always give the best results.

The system can now be modified again to include the application of those tricks. Let $I \subset [\![1, n]\!]$ be a set of cardinality $2^\gamma a_0$ such that I is the union of a_0 disjoint dyadic blocks. The system becomes:

$$\left(\left(\left(\boldsymbol{I}_d \quad | \quad \boldsymbol{U}\right) \cdot \text{Short}_I\left(\boldsymbol{G}_{\text{inv}}\right)\right) \star \text{Punct}_I\left(\boldsymbol{H}_{\text{pub}}\right)\right) \cdot \text{Punct}_I \begin{pmatrix} 0 \\ 1 \\ 0 + A_2 \\ \vdots \\ T_{n_0} \\ \vdots \\ T_{n_0} + \sum_{i=1}^{\gamma} A_i \end{pmatrix} = 0$$

where $d = \dim \text{Short}_I\left(\mathscr{D}\right) = k_0 - c - a_0$.

For the different versions of DAGS, we obtain the following reduction of variables:

Table 5.2: Reduction of the variables

	Initial $\#U$ $(k_0 - c)\, c$	Initial $\#X$ n	Total	Final $\#U$ $(k_0 - a_0 - c)\, c$	Final $\#X$ $n_0 - a_0 + \gamma - 2$	Total
DAGS_1	88	832	920	$88 - 4a_0$	$54 - a_0$	$142 - 5a_0$
DAGS_3	48	1216	1264	$48 - 4a_0$	$41 - a_0$	$89 - 5a_0$
DAGS_5	18	1600	1618	$18 - 2a_0$	$37 - a_0$	$55 - 3a_0$

The variable a_0 is the number of indices that we shorten with the last trick presented. The numerical results that are presented afterwards are using all these tricks, and the next chapters as well.

5.3.2.3 Finishing the attack

Both of the versions explained imply that we have the knowledge of $\mathscr{NT}\left(\boldsymbol{x}\right)$ or $\mathscr{NT}\left(\boldsymbol{x}_{\mathcal{I}}\right)$. Thus, it is possible to apply the same steps to recover the private vectors $\boldsymbol{x}$ and $\boldsymbol{y}$:

1. **Recover $\boldsymbol{x}$ from $\mathscr{NT}(\boldsymbol{x})$ or $\boldsymbol{x}_{\mathcal{I}}$ from $\mathscr{NT}(\boldsymbol{x}_{\mathcal{I}})$**
   Let's consider we have found the $\mathscr{NT}(\boldsymbol{x})$. This code has a dimension 4 over $\mathbb{F}_q$ and we can prove that:

$$\mathscr{NT}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2} = \langle \mathbb{1}, \boldsymbol{x}, \boldsymbol{x}^q, \boldsymbol{x}^{(q+1)} \rangle$$

   Thanks to the 2-transitivity of the affine group on $\mathbb{F}_{q^2}$, we can specialize the value of the two first entries: we will consider the first entry of $\boldsymbol{x}$ to be 0, and the second to be 1. Then, we shorten $\mathscr{NT}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2}$. We get the following code:

$$\mathscr{S} = \mathrm{Short}_{\{1\}}\left(\mathscr{NT}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2}\right) = \langle \boldsymbol{x}, \boldsymbol{x}^q, \boldsymbol{x}^{q+1} \rangle$$

   We can deduce the intersection between the code $\mathscr{S}$ and its square:

$$\mathscr{S} \cap \mathscr{S}^2 = \langle \boldsymbol{x}^{q+1} \rangle$$

   The second entry of $\boldsymbol{x}$ being 1, we deduce the value of $\boldsymbol{x}^{(q+1)}$. To find $\boldsymbol{x}$, we then need to enumerate the vectors of $\mathscr{NT}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2}$ that have 0 and 1 as their first and second entries. For any vector $\boldsymbol{c}$ that fills that condition, we compute $\boldsymbol{c}^{q+1}$. We compare this value to the value of $\boldsymbol{x}^{q+1}$, if they are equal then $\boldsymbol{c}$ is either $\boldsymbol{x}$ or $\boldsymbol{x}^q$.

   Remark 5.16. We can actually make the computations faster, by trying to recover $\left(\mathrm{Tr}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2}\right) = \langle \mathrm{Tr}(\boldsymbol{x}), \mathrm{Tr}(\alpha\boldsymbol{x}) \rangle_{\mathbb{F}_{q^2}}$. We don't need to shorten the code anymore, and we get:

$$\mathrm{Tr}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2} = \langle \boldsymbol{x}, \boldsymbol{x}^q \rangle_{\mathbb{F}_{q^2}}$$

   Then, we have $\left(\mathrm{Tr}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2}\right)^2 \cap \left(\mathscr{NT}(\boldsymbol{x}) \otimes \mathbb{F}_{q^2}\right) = \langle x^{q+1} \rangle$

   We can still wonder: how can we find $\mathrm{Tr}(\boldsymbol{x})$ and $\mathrm{Tr}(\alpha\boldsymbol{x})$? The first way to do so is explained in Barelli's thesis [15]. We will need to solve two systems in the form of (5.6), where we are searching the value of $\mathrm{Tr}(\boldsymbol{x})$ for the first system, an $\mathrm{Tr}(\alpha\boldsymbol{x})$ for the second one. Both those systems have the same first entry x_1 of $\boldsymbol{x}$ which is equal to 0 due to the 2-transitivity: we have $\mathrm{Tr}(x_1) = \mathrm{Tr}(t_1) = 0$, and we can deduce that T_1 can be replaced by 0. The two different systems actually differ by the value that replaces the variable A_1. The second entry x_2 of $\boldsymbol{x}$ is equal to 1: for the first system, we have $\mathrm{Tr}(a_1) = 0$, which means that we can replace A_1 by 0; for the second system, we have $\mathrm{Tr}(\alpha a_1) = \mathrm{Tr}(\alpha) = 1$, thus we can replace A_1 by 1. This way works, but it requires to do two Gröbner bases, which increases the complexity of performing the attack.

   There is another way of obtaining the values of $\mathrm{Tr}(\boldsymbol{x})$ and $\mathrm{Tr}(\alpha\boldsymbol{x})$, and it only requires one Gröbner basis computation. We specialize the variables T_1 and A_1 to respectively 0 and 0, and the variable A_2 to 1. We can do this last specialization thanks to the fact that we are searching for the

value of a trace. With these values, we solve the system and compute the corresponding conductor. To get the values of $\mathrm{Tr}(\boldsymbol{x})$ and $\mathrm{Tr}(\alpha\boldsymbol{x})$, we use the generator matrix of the conductor. For $\mathrm{Tr}(\boldsymbol{x})$, we add its third and fourth columns, thus obtaining a vector of the form $\begin{pmatrix}0 & 0 & 1 & 1 & \dots\end{pmatrix}$. For $\mathrm{Tr}(\alpha\boldsymbol{x})$, we add its second and fourth columns, resulting in a vector of the form $\begin{pmatrix}0 & 1 & 0 & 1 & \dots\end{pmatrix}$.

We may encounter some cases where this reasoning does not work, when values of A_i are in the base field instead of the extension, because the trace of such values is equal to 0. We then need to tweak the algorithm to find the correct trace vectors, but it is still possible.

2. **Recover $\boldsymbol{y}$ from $\boldsymbol{x}$ or $\boldsymbol{y}_{\mathcal{I}}$ from $\boldsymbol{x}_{\mathcal{I}}$**
We know that the public code $\mathscr{C}_{\mathrm{pub}}$ is alternant, so its parity-check matrix $\boldsymbol{H}_{\mathrm{pub}}$ follows the properties and form shown in 4.2.3.2. Let $\boldsymbol{G}_{\mathrm{pub}}$ be the generator matrix of $\mathscr{C}_{\mathrm{pub}}$. Since the coordinates of $\boldsymbol{x}$ (resp. $\boldsymbol{x}_{\mathcal{I}}$) are known from the previous step, we can compute the coordinates of $\boldsymbol{y}$ (resp. $\boldsymbol{y}_{\mathcal{I}}$) by solving the linear system:

$$\boldsymbol{H}(\boldsymbol{x},\boldsymbol{y}) \cdot \boldsymbol{G}_{\mathrm{pub}}^{T} = 0 \tag{5.9}$$

3. **If necessary, recover $\boldsymbol{x}$ and $\boldsymbol{y}$ from $\boldsymbol{x}_{\mathcal{I}}$ and $\boldsymbol{y}_{\mathcal{I}}$**
This step is necessary only if we made the computations after shortening the public code $\mathscr{C}_{\mathrm{pub}}$. We consider $\mathcal{I} = \{s, s+1, \dots, n\}$ the set of the indices that were shortened. We suppose here that they are the indices of the last columns; if otherwise, we can use an appropriate reordering. From the previous steps, we have the knowledge of the coordinates $x_1, \dots, x_{s-1}$ and $y_1, \dots, y_{s-1}$. We begin by setting $\mathcal{I}' = \mathcal{I} \setminus \{s\}$. Let $\boldsymbol{G}(\mathcal{I}')$ be a generator matrix for the code $\mathscr{A}_r(\boldsymbol{x}_{\mathcal{I}'}, \boldsymbol{y}_{\mathcal{I}'})$. We can write the following equation:

$$\begin{pmatrix} y_1 & \dots & y_s \\ x_1 y_1 & \dots & x_s y_s \\ \vdots & & \vdots \\ x_1^{r-1} y_1 & \dots & x_s^{r-1} y_s \end{pmatrix} \cdot \boldsymbol{G}(\mathcal{I}') = 0 \tag{5.10}$$

The only unknowns here are x_s and y_s.

We can compute y_s by solving the linear system:

$$\begin{pmatrix} y_1 & \dots & y_s \end{pmatrix} \cdot \boldsymbol{G}(\mathcal{I}') = 0 \tag{5.11}$$

Once we have y_s, we can compute x_s by solving the linear system:

$$\begin{pmatrix} x_1 y_1 & \dots & x_s y_s \end{pmatrix} \cdot \boldsymbol{G}(\mathcal{I}') = 0 \tag{5.12}$$

We can iterate this reasoning to compute the next entries until we recover the whole vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

5.3.3 Complexity results

Here we will present the complexities depending on the type of attack we chose to do. We will analyse what we can see in the tables in order to understand the computations that were made.

5.3.3.1 Combinatorial Attack Complexity

For the combinatorial version of the attack, we have the following estimated complexities:

Table 5.3: Numerical complexity of the combinatorial attack

DAGS version	Claimed Security	Approximate Complexity
DAGS_1	128 bits	$\approx 2^{70}$
DAGS_3	192 bits	$\approx 2^{80}$
DAGS_5	256 bits	$\approx 2^{58}$

To get those complexities, we used the formula (5.5) and replaced the variables by the values presented in Table 5.1.

We notice that they are clearly under the security level they claimed to be at first. The number are still high to do the computations in practice, but the attack is considered to work nonetheless. Another thing that we can see from this table is that, although the claimed security and the sizes of the vectors used are the highest, DAGS_5 actually have the lowest complexity for this attack. This can be explained by the choices that were made for the parameters q and γ. Indeed, the complexity has an exponent of $\frac{4q}{|\mathcal{G}|}$, which, computed, is 8 for DAGS_1 and DAGS_3 but only 4 for DAGS_5.

5.3.3.2 Algebraic Attack Complexity

For algebraic attacks, it is much harder to estimate the complexity so the complexity presented are only practical. Here are the approximated times for the attack on each version of DAGS, computed as an average from multiple runs:

Table 5.4: Average time for the algebraic attack in [16]

DAGS version	Claimed Security	Approximate Time
DAGS_1	128 bits	19 min
DAGS_3	192 bits	–
DAGS_5	256 bits	< 1 min

On this table, we can see that DAGS_1 can be attacked successfully in a reasonable amount of time, whereas the attack on DAGS_3 was not yet achieved. Indeed, a choice was made in order to reduce the number of variables to its

minimum, but in the case of DAGS_3, it resulted in the practical impossibility to realize the attack correctly. This issue will be discussed in detail in the next chapter. We also notice that, just as it was the case for the combinatorial attack, DAGS_5 seems to be the easiest to attack. The explanation is the same, it is due to the choice of parameters.

5.3.4 Consequences on parameters

Due to the attack in [16], the authors updated their sets of parameters, as shown in Table 5.5. We can notice two major changes in the values : the sizes q of the finite fields have been increased for all sets of parameters, and the code-related values have been modified. More precisely, the length n has been reduced from 2112 to 1600, while the dimension k has been raised from 704 to 896.

Table 5.5: DAGS updated sets of parameters

	q	m	n	k	γ	t	NIST Security Level
DAGS_1	2^6	2	832	416	4	13	Level 1 (128 bits AES)
DAGS_3	2^8	2	1216	512	5	11	Level 3 (192 bits AES)
DAGS_5	2^8	2	1600	896	5	11	Level 5 (256 bits AES)

We can directly notice the effect of those changes when calculating the complexity of the combinatorial version of the attack. They are shown in the table 5.6. We can see that DAGS_3 and DAGS_5 are completely exceeding their respective claimed security. It is slightly different fore DAGS_1 : it is close to the claimed security.

Table 5.6: Complexity of the combinatorial attack for updated parameters

DAGS version	Claimed Security	Approximate Complexity
DAGS_1	128 bits	$\approx 2^{127}$
DAGS_3	192 bits	$\approx 2^{287}$
DAGS_5	256 bits	$\approx 2^{289}$

Chapter 6

A MinRank Attack on DAGS

Contents

6.1 Introduction

In the previous chapter we saw two different versions of the attack, one being combinatorial and the other algebraic. Although the complexity of the combinatorial one is known more precisely, the algebraic one is more efficient and takes less time to succeed. We focus on the algebraic version, that worked initially only on the sets of parameters DAGS_1 and DAGS_5.

Our goal here will be to make it work on the last set of parameters, DAGS_3, and to shorten the time taken to attack the other sets of parameters. We begin by making some changes in the modeling, either by changing the inputs or by rewriting the system so that we can apply the method seen in Chapter 3.

This leads to the application of the MinRank attack on the DAGS cryptosystem, which is currently the best alternative. It works efficiently for the three first sets of parameters.

The authors of DAGS published updated parameters, and the attacks we did would not be effective against them. We decided on relying on an hybrid attack that mixes brute force search and algebraic modeling to try to attack those new parameters.

Outline of the chapter

In this chapter, we first present multiples changes for the modeling in 6.2, whether they are about the inputs or with the matrix $\boldsymbol{B}$. We then describe how the MinRank attack adapts to the DAGS cryptosystem in 6.3. We give the results of all those previous explained methods in 6.4. Finally, in 6.5, we present an alternative way of attacking the new parameters.

6.2 Modifying the Modeling

After the publishing of the article [16], we tried to see if we could find any way of improving further the attack, either by making it faster or by managing to attack the DAGS_3 version. We found multiple improvements that reduced considerably the computations time, and, that allowed us to successfully attack DAGS_3. We will present here these improvements one after another.

Before introducing those improvements, we recall the formula of the system:

$$\left(\left(\left(\boldsymbol{I}_d \quad | \quad \boldsymbol{U}\right) \cdot \mathrm{Short}_I\left(\boldsymbol{G}_{\mathrm{inv}}\right)\right) \star \mathrm{Punct}_I\left(\boldsymbol{H}_{\mathrm{pub}}\right)\right) \cdot \begin{pmatrix} 0 \\ 1 \\ 0 + b_2 \\ \vdots \\ \tau_{n_0 - a_0} \\ \vdots \\ \tau_{n_0 - a_0} + \sum_{i=1}^{\gamma} b_i \end{pmatrix} = 0$$

with

- $\boldsymbol{I}_d$ the identity matrix of size $d \times d$
- $\boldsymbol{U}$ the matrix of size $(k_0 - a_0 - c) \times c$ of variables
- $\boldsymbol{G}_{\mathrm{inv}}$ the generator matrix of the invariant subcode of the public code
- $\boldsymbol{H}_{\mathrm{pub}}$ the parity-check matrix of the public code
- $\tau_1, \ldots, \tau_{n_0 - a_0}$ the $n_0 - a_0$ variables on which we apply the permutation group
- $b_1, \ldots, b_\gamma$ the γ unknown elements of the permutation group

We can easily identify the two sets of variables that make this system bilinear: one is the variables from $\boldsymbol{U}$ the other corresponds to both variables τ and b.

To ease the writing, from now on, we will not specify the shortening and puncturing and when relevant, we will call $\boldsymbol{V}$ the vector made from the variables τ and b. This leads to the system being rewritten as:

$$\left(\left(\left(\boldsymbol{I}_d \quad | \quad \boldsymbol{U}\right) \cdot \boldsymbol{G}_{\mathrm{inv}}\right) \star \boldsymbol{H}_{\mathrm{pub}}\right) \cdot \boldsymbol{V}^T = 0$$

6.2.1 Eliminating redundancy among the equations

The first improvement came from the observation that during the computations we had a huge number of equations that seemed to be almost all reduced after a few steps. It was explained in [10].

This behavior comes from the fact that $\boldsymbol{G}_{\mathrm{inv}}$ and $\boldsymbol{H}_{\mathrm{pub}}$ are not independent from each other. Indeed, $\boldsymbol{G}_{\mathrm{inv}}$ is the generator matrix of the invariant

code associated to the code $\mathscr{C}_{\text{pub}}$, whose $\boldsymbol{H}_{\text{pub}}$ is a parity-check matrix. Their structure and the fact that we do a product with the vector $\boldsymbol{V}$ that has every entries repeated (due to the fact that $b_1 = 0$) creates redundancy in the final polynomials that we obtain.

Let's take two rows $\boldsymbol{r}$ and $\boldsymbol{r}^\sigma$ from $\boldsymbol{H}_{\text{pub}}$ in the same quasi-dyadic block where σ is a permutation in $\mathcal{G}$. For any row $\boldsymbol{u} = \boldsymbol{u}^\sigma$ from the invariant matrix $\begin{pmatrix}\boldsymbol{I}_d & \boldsymbol{U}\end{pmatrix}\mathcal{S}_I\boldsymbol{G}_{\text{inv}}$, the component-wise product with $\boldsymbol{r}^\sigma$ satisfies $\boldsymbol{u}\star\boldsymbol{c}^\sigma = (\boldsymbol{u}\star\boldsymbol{c})^\sigma$. As $\boldsymbol{V} = \boldsymbol{V}^\sigma + \sigma\mathbb{1}_n$ and $\boldsymbol{u}\star\boldsymbol{c}\cdot\mathbb{1}_n = 0$, we have a redundancy with as many repetition as the size of the permutation group $\mathcal{G}$.

This problem does not have an impact on the fact that we can find a solution, but it may slow down the computation of the Gröbner basis, as the computer will have to deal with more polynomials than what is actually needed. It is however a problem to accurately estimate the complexity of computations. We can solve this problem by deliberately picking 1 row every 2^γ in the matrix $\boldsymbol{H}$.

6.2.2 Altering the number of rows of $\boldsymbol{U}$

This last improvement is the one that was the most effective in reducing the practical complexity of the attack. In the previous chapter, we addressed the fact that intuitively, we want to have the minimum number of variables possible. We also noticed that reasoning like that did not allow DAGS_3 to be attacked. We studied the DAGS system and identified that the number of variables was heavily influenced by the number of rows of the matrix $\boldsymbol{U}$.

This number of rows of the matrix $\boldsymbol{U}$ can vary between 2 and $k_0 - a_0 - c$. 2 is the minimum to be able to perform the attack but in some cases, as it is for DAGS_3, the ratio between equations and variables is too small for the attack to be doable practically. To actually being able to do the attack, we have to consider keeping more rows from the matrix $\boldsymbol{U}$. Indeed, this means keeping more variables but also more equations. In the case of DAGS, for 1 row of $\boldsymbol{U}$ that we keep, we get more equations than variables. The size of the identity matrix concatenated to $\boldsymbol{U}$ corresponds to the number of rows we decide on for $\boldsymbol{U}$. The details of how many of each we keep for 1 row depending on the set of parameters we are considering is given in the table 6.2.

6.2.3 Rewriting the DAGS Attack System

We recall the system we have for the attack on DAGS from previous chapters:

$$\left(\left(\begin{pmatrix}\boldsymbol{I}_d & \boldsymbol{U}\end{pmatrix}\cdot\boldsymbol{G}_{\text{inv}}\right)\star\boldsymbol{H}_{\text{pub}}\right)\cdot\boldsymbol{V}^T = 0$$

Unfortunately, this form does not allow us to apply easily the method presented in section 3.2. Thus, we need to transform it into a product of matrices instead of a mix of classical and star products. To help with doing such thing, we need to introduce two results about equalities between different matricial products. The proof of the two lemmas can be found in the appendix A.

Lemma 6.1. *Let $\boldsymbol{v}$ a vector of size $m_{\boldsymbol{v}}$ and $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two matrices with $m_{\boldsymbol{v}}$ columns. We can write:*

$$(\boldsymbol{X} \star \boldsymbol{v}) \cdot \boldsymbol{Y}^T = \boldsymbol{X} \cdot (\boldsymbol{Y} \star \boldsymbol{v})^T \tag{6.1}$$

$$(\boldsymbol{X} \star \boldsymbol{Y})\, \boldsymbol{v}^T = \mathrm{vec}_{row}\left((\boldsymbol{X} \star \boldsymbol{v})\, \boldsymbol{Y}^T\right) \tag{6.2}$$

Lemma 6.2. *Let $\boldsymbol{A}$ and $\boldsymbol{C}$ two matrices with the same number of columns, and $\boldsymbol{b}$ and $\boldsymbol{d}$ two vectors of the same size.*

$$(\boldsymbol{A} \otimes \boldsymbol{b}) \star (\boldsymbol{C} \otimes \boldsymbol{d}) = (\boldsymbol{A} \star \boldsymbol{C}) \otimes (\boldsymbol{b} \star \boldsymbol{d}) \tag{6.3}$$

Proposition 6.3 (Kronecker product properties)**.** *Let $\boldsymbol{A}$ and $\boldsymbol{C}$ two matrices of the same size, and $\boldsymbol{B}$ and $\boldsymbol{D}$ two matrices also of the same size.*

- *Mixed-product property*

$$(\boldsymbol{A} \otimes \boldsymbol{B}) \cdot (\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{A} \cdot \boldsymbol{C}) \otimes (\boldsymbol{B} \cdot \boldsymbol{D}) \tag{6.4}$$

- *Transpose property*

$$(\boldsymbol{A} \otimes \boldsymbol{B})^T = \boldsymbol{A}^T \otimes \boldsymbol{B}^T \tag{6.5}$$

We can now use these two equations to transform our system and get it into a better form to work with. In order to do that, we have to define new elements:

- $\boldsymbol{G}_{\mathrm{f}}$ is the matrix such that $\boldsymbol{G}_{\mathrm{inv}} = \begin{pmatrix}\boldsymbol{I}_{k_0} & \boldsymbol{G}_{\mathrm{f}}\end{pmatrix} \cdot (\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma})$
- $\overline{\boldsymbol{H}}$ corresponds to the matrix $\boldsymbol{H}_{\mathrm{pub}}$ where we take only 1 row every 2^γ.
- $\boldsymbol{H}_{\mathrm{i}}$ is the matrix we define as $\boldsymbol{H}_{\mathrm{i}} = \overline{\boldsymbol{H}} \cdot \left(\boldsymbol{I}_{n_0} \otimes \hat{\boldsymbol{e}}_i^T\right)$ with $\hat{\boldsymbol{e}}_i$ the vector of 0 and 1 such that $\hat{\boldsymbol{e}}_i = \mathbb{1}_{2^{\gamma-i}} \otimes \begin{pmatrix}0 & 1\end{pmatrix} \otimes \mathbb{1}_{2^{i-1}}$

Theorem 6.4. *The DAGS attack system can be rewritten as:*

$$\left(\sum_{i=1}^{k_0} \tau_i \begin{pmatrix}\mathbf{0} & \left(-\boldsymbol{G}_{\mathrm{f}}^T\right)_{[*,i]} & \mathbf{0}\end{pmatrix} + \sum_{i=1}^{n_0-k_0-1} \tau_{k_0+i} \begin{pmatrix}\mathbf{0} \\ \left(\boldsymbol{G}_{\mathrm{f}}^T\right)_{[i,*]} \\ \mathbf{0}\end{pmatrix} + \sum_{i=1}^{\gamma} b_i \left(\boldsymbol{H}_{\mathrm{i}}\right)_{[*,\{1\ldots k_0\}]}\right) \cdot \begin{pmatrix}\boldsymbol{I}_{k_0-c} \\ \boldsymbol{U}^T\end{pmatrix} = \mathbf{0}$$

Proof. We begin by directly applying Lemma 6.1 on the system:

$$\left(\left(\begin{pmatrix}\boldsymbol{I}_d & \boldsymbol{U}\end{pmatrix} \cdot \boldsymbol{G}_{\mathrm{inv}}\right) \star \boldsymbol{H}_{\mathrm{pub}}\right) \cdot \boldsymbol{V}^T = \left(\begin{pmatrix}\boldsymbol{I}_d & \boldsymbol{U}\end{pmatrix} \cdot \boldsymbol{G}_{\mathrm{inv}}\right) \left(\overline{\boldsymbol{H}} \star \boldsymbol{V}\right)^T$$

$\boldsymbol{G}_{\mathrm{inv}}$ is the generator matrix of the invariant code associated with the code whose $\boldsymbol{H}_{\mathrm{pub}}$ is the parity-check matrix, it can be expressed as $\boldsymbol{G}'_{\mathrm{f}} \cdot (\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma})$

where $\boldsymbol{G}'_{\mathrm{f}} = \begin{pmatrix} \boldsymbol{I}_{k_0} & \boldsymbol{G}_{\mathrm{f}} \end{pmatrix}$. The matrix $\overline{\boldsymbol{H}}$ corresponds to the matrix $\boldsymbol{H}_{\mathrm{pub}}$ where we take only 1 row every 2^γ. We can directly rewrite the previous equation as:

$$\begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{U} \end{pmatrix} \cdot \boldsymbol{G}'_{\mathrm{f}} \cdot \left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right) \cdot \left(\overline{\boldsymbol{H}} \star \boldsymbol{V}\right)^T = \boldsymbol{0}_{(k_0-c)\times(n_0-k_0)}$$
$$\begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{U} \end{pmatrix} \cdot \boldsymbol{G}'_{\mathrm{f}} \cdot \left(\left(\overline{\boldsymbol{H}} \star \boldsymbol{V}\right)\left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)^T\right)^T = \boldsymbol{0}_{(k_0-c)\times(n_0-k_0)}$$

Now, we explain how we modify a subpart of the formula of the system:

$$\begin{aligned}
&\left(\overline{\boldsymbol{H}} \star \boldsymbol{V}\right)\left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)^T \\
=&\left(\overline{\boldsymbol{H}} \star \left(\vec{\tau} \otimes \mathbb{1}_{2^\gamma} + \mathbb{1}_{n_0} \otimes \sum_{i=1}^{\gamma} b_i \hat{\boldsymbol{e}}_i\right)\right)\left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)^T
\end{aligned}$$

We develop the star product, then the classical product of 2 matrices:

$$\begin{aligned}
=&\left(\overline{\boldsymbol{H}} \star \left(\vec{\tau} \otimes \mathbb{1}_{2^\gamma}\right) + \overline{\boldsymbol{H}} \star \left(\mathbb{1}_{n_0} \otimes \sum_{i=1}^{\gamma} b_i \hat{\boldsymbol{e}}_i\right)\right)\left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)^T \\
=&\left(\overline{\boldsymbol{H}} \star \left(\vec{\tau} \otimes \mathbb{1}_{2^\gamma}\right)\right)\left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)^T + \left(\overline{\boldsymbol{H}} \star \left(\mathbb{1}_{n_0} \otimes \sum_{i=1}^{\gamma} b_i \hat{\boldsymbol{e}}_i\right)\right)\left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)^T
\end{aligned}$$

We apply Lemma 6.1:

$$=\overline{\boldsymbol{H}} \star \left(\left(\vec{\tau} \otimes \mathbb{1}_{2^\gamma}\right) \star \left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)\right)^T + \overline{\boldsymbol{H}} \star \left(\left(\mathbb{1}_{n_0} \otimes \sum_{i=1}^{\gamma} b_i \hat{\boldsymbol{e}}_i\right) \star \left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}\right)\right)^T$$

We now apply Lemma 6.2:

$$=\overline{\boldsymbol{H}} \star \left(\left(\boldsymbol{I}_{n_0} \star \boldsymbol{\tau}\right) \otimes \left(\mathbb{1}_{2^\gamma} \star \mathbb{1}_{2^\gamma}\right)\right)^T + \sum_{i=1}^{\gamma} b_i \overline{\boldsymbol{H}} \left(\left(\boldsymbol{I}_{n_0} \star \mathbb{1}_{n_0}\right) \otimes \left(\mathbb{1}_{2^\gamma} \star \hat{\boldsymbol{e}}_i\right)\right)^T$$

We develop the transpose for both the Kronecker products as in Equation 6.5:

$$=\overline{\boldsymbol{H}} \star \left(\mathrm{diag}\left(\boldsymbol{\tau}\right)^T \otimes \mathbb{1}_{2^\gamma}^T\right) + \sum_{i=1}^{\gamma} b_i \overline{\boldsymbol{H}} \left(\boldsymbol{I}_{n_0}^T \otimes \hat{\boldsymbol{e}}_i^T\right)$$

Finally, we apply Equation 6.4:

$$\begin{aligned}
=&\overline{\boldsymbol{H}} \star \left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}^T\right)\left(\mathrm{diag}\left(\boldsymbol{\tau}\right) \otimes 1\right) + \sum_{i=1}^{\gamma} b_i \overline{\boldsymbol{H}} \left(\boldsymbol{I}_{n_0}^T \otimes \hat{\boldsymbol{e}}_i^T\right) \\
=&\boldsymbol{H}_{\mathrm{f}} \cdot \mathrm{diag}\left(\boldsymbol{\tau}\right) + \sum_{i=1}^{\gamma} b_i \boldsymbol{H}_i
\end{aligned}$$

where $\boldsymbol{H}_{\mathrm{f}} = \overline{\boldsymbol{H}} \cdot \left(\boldsymbol{I}_{n_0} \otimes \mathbb{1}_{2^\gamma}^T\right)$ and $\boldsymbol{H}_i = \overline{\boldsymbol{H}} \cdot \left(\boldsymbol{I}_{n_0} \otimes \hat{\boldsymbol{e}}_i^T\right)$.

Our complete system has now become:

$$\begin{pmatrix}\boldsymbol{I}_d & \boldsymbol{U}\end{pmatrix}\cdot\boldsymbol{G}'_{\mathrm{f}}\cdot\left(\boldsymbol{H}_{\mathrm{f}}\cdot\operatorname{diag}(\boldsymbol{\tau})+\sum_{i=1}^{\gamma}b_i\boldsymbol{H}_i\right)^T=\boldsymbol{0}_{(k_0-c)\times(n_0-k_0)}$$

We continue the transformation to obtain a system with the most suitable form by applying a transpose on both sides of the equation:

$$\left(\boldsymbol{H}_{\mathrm{f}}\cdot\operatorname{diag}(\boldsymbol{\tau})\cdot\boldsymbol{G}'^T_{\mathrm{f}}+\sum_{i=1}^{\gamma}b_i\boldsymbol{H}_i\cdot\boldsymbol{G}'^T_{\mathrm{f}}\right)\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \boldsymbol{U}^T\end{pmatrix}=\boldsymbol{0}_{(n_0-k_0)\times(k_0-c)}$$

To go further, we need to recall the definitions of some matrices:

$$\begin{aligned}\boldsymbol{H}_{\mathrm{f}}&=\begin{pmatrix}-\boldsymbol{G}_{\mathrm{f}}^T & \boldsymbol{I}_{n_0-k_0}\end{pmatrix}\\ \boldsymbol{H}_i&=\begin{pmatrix}(\boldsymbol{H}_i)_{[*,\{1\dots k_0\}]} & \boldsymbol{0}\end{pmatrix}\\ \boldsymbol{G}'_{\mathrm{f}}&=\begin{pmatrix}\boldsymbol{I}_{k_0} & \boldsymbol{G}_{\mathrm{f}}\end{pmatrix}\\ \operatorname{diag}(\boldsymbol{\tau})&=\begin{pmatrix}\operatorname{diag}(\tau_1,\dots,\tau_{k_0}) & \boldsymbol{0}\\ \boldsymbol{0} & \operatorname{diag}(\tau_{k_0+1},\dots,\tau_{n_0-1},0)\end{pmatrix}\end{aligned}$$

With the simplifications induced by the way we can write those matrices, our system becomes:

$$\begin{aligned}&\Big(-\boldsymbol{G}_{\mathrm{f}}^T\operatorname{diag}(\tau_1\dots\tau_{k_0})+\operatorname{diag}(\tau_{k_0+1}\dots\tau_{n_0-1},0)\,\boldsymbol{G}_{\mathrm{f}}^T\\ &\qquad+\sum_{i=1}^{\gamma}b_i\,(\boldsymbol{H}_i)_{[*,\{1\dots k_0\}]}\Big)\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \boldsymbol{U}^T\end{pmatrix}=\boldsymbol{0}_{(n_0-k_0)\times(k_0-c)}\\ &\left(\sum_{i=1}^{k_0}\tau_i\begin{pmatrix}\boldsymbol{0} & \left(-\boldsymbol{G}_{\mathrm{f}}^T\right)_{[*,i]}\\ & \boldsymbol{0}\end{pmatrix}+\sum_{i=1}^{n_0-k_0-1}\tau_{k_0+i}\begin{pmatrix}\boldsymbol{0}\\ \left(\boldsymbol{G}_{\mathrm{f}}^T\right)_{[i,*]}\\ \boldsymbol{0}\end{pmatrix}\right.\\ &\qquad\left.+\sum_{i=1}^{\gamma}b_i\,(\boldsymbol{H}_i)_{[*,\{1\dots k_0\}]}\right)\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \boldsymbol{U}^T\end{pmatrix}=\boldsymbol{0}_{(n_0-k_0)\times(k_0-c)}\end{aligned}$$

□

To be able to apply the reasoning seen before, we still need to separate the homogeneous and linear parts. We can do that by dividing $\boldsymbol{G}_{\mathrm{f}}$ into two parts $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ such that $\boldsymbol{G}_{\mathrm{f}}=\begin{pmatrix}\boldsymbol{G}_1\\ \boldsymbol{G}_2\end{pmatrix}$.

The *homogeneous part of highest degree* comes from the product between the unknowns of the matrix $\boldsymbol{U}$ and those of the submatrix $\boldsymbol{G}_2$ of $\boldsymbol{G}_{\mathrm{f}}$.

$$\left(\sum_{i=1}^{c}\tau_{k_0-c+i}\begin{pmatrix}\boldsymbol{0} & \left(-\boldsymbol{G}_2^T\right)_{[*,i]} & \boldsymbol{0}\end{pmatrix}+\sum_{i=1}^{n_0-k_0-1}\tau_{k_0+i}\begin{pmatrix}\boldsymbol{0}\\ \left(\boldsymbol{G}_2^T\right)_{[i,*]}\\ \boldsymbol{0}\end{pmatrix}\right.$$

$$+\sum_{i=3}^{\gamma} b_i \left(\boldsymbol{H}_i\right)_{[*,\{k_0-c+1\ldots k_0\}]}\Bigg) \cdot \boldsymbol{U}^T$$

The *linear and constant parts* are corresponding to multiple products that include the identity matrix of $\begin{pmatrix}\boldsymbol{I}_d & \boldsymbol{U}\end{pmatrix}$ or the matrix $\boldsymbol{H}_2$ that is known.

$$\sum_{i=1}^{k_0-c} \tau_i \begin{pmatrix}\mathbf{0} & \left(-\boldsymbol{G}_1^T\right)_{[*,i]} & \mathbf{0}\end{pmatrix} + \sum_{i=1}^{n_0-k_0-1} \tau_{k_0+i} \begin{pmatrix}\mathbf{0} \\ \left(\boldsymbol{G}_1^T\right)_{[i,*]} \\ \mathbf{0}\end{pmatrix}$$

$$+\sum_{i=3}^{\gamma} b_i \left(\boldsymbol{H}_i\right)_{[*,\{1\ldots k_0-c\}]} + \left(\boldsymbol{H}_2\right)_{[*,\{k_0-c+1\ldots k_0\}]} \cdot \boldsymbol{U}^T + \left(\boldsymbol{H}_2\right)_{[*,\{1\ldots k_0-c\}]}$$

This way of writing the system is the one we will use to be able to apply the methodology described in Section 3.2.

As we saw in Chapter 3, using minors with τ and b variables does not improve the computations. We will then focus on the U variables.

6.2.4 U variables

We can now consider the other jacobian matrix, with respect to the τ and b variables, which is the one that will bring us better results. Indeed, the solving of the system using this U variables gets degree falls, when they exist, at a lower degree. To make the reading easier, we identify the matrices of this system to the matrices $\boldsymbol{M}_i$ we saw before:

$$M_i = \begin{cases} \mathbf{0} & \text{for } i \in [1,\ldots,k_0-c] \\ \begin{pmatrix}\mathbf{0} & \left(-\boldsymbol{G}_2^T\right)_{[*,i-k_0+c]} & \mathbf{0}\end{pmatrix} & \text{for } i \in [k_0-c+1,\ldots,k_0] \\ \begin{pmatrix}\mathbf{0} \\ \left(\boldsymbol{G}_2^T\right)_{[i-k_0,*]} \\ \mathbf{0}\end{pmatrix} & \text{for } i \in [k_0+1,\ldots,n_0-1] \\ \left(\boldsymbol{H}_{i-n_0+1}\right)_{[*,\{k_0-c+1\ldots k_0\}]} & \text{for } i \in [n_0,\ldots,n_0-1+\gamma-2] \end{cases}$$

Due to the form of the homogeneous part, we can calculate separately the jacobian for each sum and we construct the complete jacobian with a concatenation of the different parts :

$$\text{jac}_{(\boldsymbol{\tau},\boldsymbol{b})}\left(\text{vec}_{row}\left(\mathcal{F}_h\right)\right) = \tag{6.6}$$
$$\left(\text{jac}_{(\tau_{k_0-c+1},\ldots,\tau_{k_0})}\left(\text{vec}_{row}\left(\mathcal{F}_h\right)\right) \middle| \text{jac}_{(\tau_{k_0+1},\ldots,\tau_{n_0-1})}\left(\text{vec}_{row}\left(\mathcal{F}_h\right)\right) \middle| \text{jac}_{\boldsymbol{b}}\left(\text{vec}_{row}\left(\mathcal{F}_h\right)\right)\right)$$

We can notice that only part of the τ variables are appearing. Although, thanks to the identification we just did, we can write the jacobian matrix with

one unique formula, using Proposition 3.5 :

$$\text{jac}_{(\boldsymbol{\tau},\boldsymbol{b})}\left(\text{vec}_{row}\left(\mathcal{F}_h\right)\right) = \left(\boldsymbol{I}_{n_0-k_0} \otimes \boldsymbol{U}\right)\left(\text{vec}_{row}\left(M_i\right)\right)_{i=\{1,\ldots,n_0-1+\gamma-2\}}$$

Following Theorem 3.8, our next step is to construct a matrix $\boldsymbol{B}$. Indeed, we remarked previously that the Macaulay matrices can be arranged as a diagonal of matrices $\boldsymbol{B}$, and to be able to linearize such a matrix, we want to compute the ranks of all matrices $\boldsymbol{B}$. We choose to construct the matrix in 2 parts : one for the τ variables, the other for the b variables. Their behavior is different: the τ part is the one that induces degree falls and more reductions than expected, while the b part has a full rank and behaves as a generic matrix. Once the analysis is done, these parts can be concatenated to form the complete $\boldsymbol{B}$ matrix.

We remind the definition of the sets of indices T and T' that we introduced in Theorem 3.8:

$$\begin{aligned} T &= (t_1, \ldots, t_d) && \text{such that } 1 \leq t_k < t_{k+1} \leq r \\ T' &= \left(t'_1, \ldots, t'_{d+1}\right) && \text{such that } 1 \leq t'_k < t'_{k+1} \leq r \end{aligned}$$

We chose to build the vectors of all possible sets of T and T': we respectively denote them $\boldsymbol{T} = \left(T_1, \ldots, T_{\binom{c}{d}}\right)$ and $\boldsymbol{T'} = \left(T'_1, \ldots, T'_{\binom{c}{d+1}}\right)$.

6.2.4.1 $\boldsymbol{B}$ for b variables

The $\boldsymbol{B}^{(\boldsymbol{b})}$ matrix, that corresponds only to the b variables, is of size $\binom{c}{d}(n_0-k_0)\times\binom{c}{d+1}(\gamma-2)$ and its nonzero blocks are defined as :

$$\boldsymbol{B}_s^{(\boldsymbol{b})} = \left(\left(\boldsymbol{H}_i\right)_{[*,k_0-c+s]}\right)_{i=\{3,\ldots,\gamma\}} \tag{6.7}$$

when $T'\backslash T = \{s\}$.

Th behavior of this matrix is comparable to any random matrix. Its expected rank is $\binom{c}{d+1}(\gamma-2)$ and is identical to its size: the matrix is of full rank. This can be verified practically.

6.2.4.2 $\boldsymbol{B}$ for τ variables

The $\boldsymbol{B}^{(\boldsymbol{\tau})}$ matrix, that is for the τ variables, is more complex, and is the concatenation of two matrices. We will see that this matrix does not have a generic behavior, as it does not have a full rank.

By definition of the matrix $\boldsymbol{B}$ given in Subsection 3.2.2.1, we can write:

$$\boldsymbol{B}_s^{(\boldsymbol{\tau})} = \left(\boldsymbol{M}_{\ell,[*,s]}\right)_{\ell=\{1\ldots n_0-1\}}$$

As there are only the variables τ_{k_0+c-1} to τ_{n_0-1} in the homogeneous part of highest degree of the system, we can restrict the matrices $\boldsymbol{M}_\ell$

$$\boldsymbol{B}_s^{(\boldsymbol{\tau})} = \left(\left(\boldsymbol{M}_{\ell,[*,s]}\right)_{\ell=\{k_0-c+1\ldots k_0\}} \quad \left(\boldsymbol{M}_{\ell,[*,s]}\right)_{\ell=\{k_0+1\ldots n_0-1\}}\right)$$

By correspondence with the original matrices, and by changing the variable ℓ to i to remove the offset in the indices, we have:

$$\boldsymbol{B}_s^{(\boldsymbol{\tau})} = \left(\begin{pmatrix} \mathbf{0} & \left(\boldsymbol{G}_2^T\right)_{[*,i]} & \mathbf{0} \end{pmatrix}_{i=s} \quad \left(\begin{pmatrix} \mathbf{0} \\ \left(\boldsymbol{G}_2^T\right)_{[i,*]} \\ \mathbf{0} \end{pmatrix}_{[*,s]} \right)_{i=\{1,\dots,n_0-k_0-1\}} \right)$$

We detail the structure of a block of this matrix in the following lemma:

Lemma 6.5. *The matrix $\boldsymbol{B}^{(\boldsymbol{\tau})}$ has a size of $\binom{c}{d}(n_0-k_0)\times\binom{c}{d+1}(n_0-k_0+c-1)$. Its nonzero blocks are defined as :*

$$\boldsymbol{B}_s^{(\boldsymbol{\tau})} = \begin{pmatrix} 0 & \dots & g_{s,1} & \dots & 0 & g_{s,1} & 0 & 0 & 0 \\ 0 & \dots & g_{s,2} & \dots & 0 & 0 & g_{s,2} & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & 0 & 0 & \ddots & 0 \\ 0 & \dots & \vdots & \dots & 0 & 0 & 0 & 0 & g_{s,n_0-k_0-1} \\ 0 & \dots & g_{s,n_0-k_0} & \dots & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

with $T'\backslash T = \{s\}$ and $g_{i,j}$ being the element of $\boldsymbol{G}_2$ on the row i and on the column j.

Proof. We consider T and T' two sets of indices, and $T'\backslash T = \{s\}$. From 6.2.3, we can directly deduce the following form for $\boldsymbol{B}_s^{(\boldsymbol{\tau})}$:

$$\boldsymbol{B}_s^{(\boldsymbol{\tau})} = \left(\begin{pmatrix} \mathbf{0} & \left(\boldsymbol{G}_2^T\right)_{[*,i]} & \mathbf{0} \end{pmatrix}_{i=s} \quad \left(\begin{pmatrix} \mathbf{0} \\ \left(\boldsymbol{G}_2^T\right)_{[i,*]} \\ \mathbf{0} \end{pmatrix}_{[*,s]} \right)_{i=\{1,\dots,n_0-k_0-1\}} \right)$$

This writing being difficult to work with, we will try to simplify it:

$$\boldsymbol{B}_s^{(\boldsymbol{\tau})} = \left(\begin{pmatrix} \mathbf{0} & \left(\boldsymbol{G}_2^T\right)_{[*,s]} & \mathbf{0} \end{pmatrix} \quad \operatorname{diag}\left(\left(\boldsymbol{G}_2^T\right)_{[i,s]} \right)_{i=\{1,\dots,n_0-k_0-1\}} \right)$$

We can even explicitly give this block:

$$\boldsymbol{B}_s^{(\boldsymbol{\tau})} = \begin{pmatrix} 0 & \dots & g_{s,1} & \dots & 0 & g_{s,1} & 0 & 0 & 0 \\ 0 & \dots & g_{s,2} & \dots & 0 & 0 & g_{s,2} & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & 0 & 0 & \ddots & 0 \\ 0 & \dots & \vdots & \dots & 0 & 0 & 0 & 0 & g_{s,n_0-k_0-1} \\ 0 & \dots & g_{s,n_0-k_0} & \dots & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

This writing will allow us to find the rank of the complete matrix more easily. □

We can notice that those blocks, when they are not full of zeros, have a specific structure. The first c columns are composed of 0 except the s^{th} which is actually the s^{th} column of $\boldsymbol{G}_2^T$. The rest of the block is composed of a diagonal matrix composed of the same column. As we are working with a characteristic 2, we do not keep the signs that would have appeared otherwise. The last row of this right part is full of zeros.

If this was a generic matrix, the rank would have been the one given in [57]: $\binom{c}{d+1}(n_0 - k_0 + c - 1)$. This is where the effects of the structure of DAGS emerge and induce changes: the rank is not the one expected.

Theorem 6.6. *We suppose that all coefficients of $\boldsymbol{B}_c$ are different from 0 and that we do not have a column of $\boldsymbol{G}_{\mathrm{f}}$ that is only zero. The rank of $\boldsymbol{B}^{(\boldsymbol{\tau})}$ is*

$$Rank(\boldsymbol{B}^{(\boldsymbol{\tau})}) = \min\left\{ \binom{c}{d}(n_0 - k_0), \binom{c-1}{d}(n_0 - k_0) + \binom{c}{d+1}d \right\} \quad (6.8)$$

Proof. We remind that $\boldsymbol{B}^{(\boldsymbol{\tau})}$ is a block matrix, whose blocks have a size of $(n_0 - k_0) \times (n_0 - k_0 + c - 1)$. They are indexed by $\boldsymbol{T} = \left(T_1, \ldots, T_{\binom{c}{d}}\right)$ for what we will call *block rows* and $\boldsymbol{T'} = \left(T'_1, \ldots, T'_{\binom{c}{d+1}}\right)$ for *block columns*. We work in a field with a characteristic 2, so we will discard the signs we will find along the computation. The simplification of the matrix $\boldsymbol{B}^{(\boldsymbol{\tau})}$ will be done in two steps, that will help us to prove its rank.

To ease the computation of the rank of $\boldsymbol{B}^{(\boldsymbol{\tau})}$, we choose an ordering on $\boldsymbol{T}$ and $\boldsymbol{T'}$ such that all the block rows and all the block columns indexed by sets containing the value c are the last ones. With such an ordering, we can divide $\boldsymbol{B}^{(\boldsymbol{\tau})}$ in 4 parts :

$$\begin{array}{l} \\ \binom{c-1}{d} \text{ sets } T \text{ with } c \notin T \\ \binom{c-1}{d-1} \text{ sets } T \text{ with } c \in T \end{array} \begin{array}{c|c} \binom{c-1}{d+1} \text{ sets } T' \text{ with } c \notin T' & \binom{c-1}{d} \text{ sets } T \cup \{c\} \\ \hline \boldsymbol{B}_{T,T'} & \operatorname{diag}(\boldsymbol{B}_c) \\ \hline 0 & \boldsymbol{B}_{T,T'} \end{array}$$

Step 1 This first step applies operations on columns in the matrix $\boldsymbol{B}^{(\boldsymbol{\tau})}$ so that its form is right for the second step. For a given block column indexed by T', we want to consider the nonzero block $\boldsymbol{B}_s$, where s is the greatest element of T'. We want to cancel its s-th column, which is the only nonzero column among the c first ones, using the diagonal part of the matrix. For the fixed block column, we can add all columns together. This is equivalent to multiply $\boldsymbol{B}^{(\boldsymbol{\tau})}$ by a matrix on the right that allows sums of columns. The effects of this product is different whether the block where we apply it is $\boldsymbol{B}_s$ or another block.

The matrix $\boldsymbol{B}_s$ becomes:

$$\boldsymbol{B}'_s = \begin{array}{c} \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & s & & & & \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & g_{1,s} & 0 & 0 \\ 0 & 0 & \vdots & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_{n_0-k_0-1,s} \\ 0 & 0 & g_{n_0-k_0,s} & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

The other matrices $\boldsymbol{B}_k$ with $k \neq s$ become:

$$\boldsymbol{B}'_k = \begin{array}{c} \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & k & s & & & \end{array} \\ \begin{pmatrix} 0 & g_{1,k} & 0 & g_{1,k} & 0 & 0 \\ 0 & \vdots & 0 & 0 & \ddots & 0 \\ 0 & g_{n_0-k_0-1,k} & 0 & 0 & 0 & g_{n_0-k_0-1,k} \\ 0 & g_{n_0-k_0,k} & g_{n_0-k_0,k} & 0 & 0 & 0 \end{pmatrix} \end{array}$$

As we consider only one block column at a time, the process works on all the block columns independently. From now on, we will write all the modified matrices with a prime.

Step 2 The second step removes the diagonal part of all the nonzero blocks in the top-left part of the matrix, using the $\boldsymbol{B}'_c$ blocks in the top-right part.

We make the hypothesis that all the coefficients coming from $\boldsymbol{B}'_c$ are different from zero. At the end, we will examine the probability of some of them being zero, and the changes this would induce.

To do so, we need to understand what columns are involved to cancel the diagonal parts of a given block column. Let's start by giving new notation:

$$\begin{aligned} T' &= \{t_1, \ldots, t_{d+1}\} \\ T_i &= T' \setminus \{t_i\} \\ T'_i &= T' \setminus \{t_i\} \cup \{c\} \\ &= T_i \cup \{c\} \end{aligned}$$

$$
\begin{array}{c}
\\
\vdots \\
T_1 \\
\vdots \\
T_2 \\
\vdots \\
\vdots \\
T_3 \\
\vdots
\end{array}
\begin{array}{c}
\begin{array}{ccc|ccccc}
T' & \dots & & \dots & T'_1 & \dots & T'_2 & \dots \\
\end{array} \\
\left(\begin{array}{ccc|ccccc}
 & & & & & & & \\
\boldsymbol{B}^{(\boldsymbol{\tau})}_{t_1} & \dots & & \dots & \boldsymbol{B}^{(\boldsymbol{\tau})}_{c} & & & \\
\vdots & & & & \vdots & & & \\
\boldsymbol{B}^{(\boldsymbol{\tau})}_{t_2} & \dots & & \dots & \dots & \dots & \boldsymbol{B}^{(\boldsymbol{\tau})}_{c} & \\
\vdots & & & & \vdots & & \vdots & \\
\hline
\vdots & & & & \vdots & & \vdots & \\
0 & \dots & & \dots & \boldsymbol{B}^{(\boldsymbol{\tau})}_{t_2} & \dots & \boldsymbol{B}^{(\boldsymbol{\tau})}_{t_1} & \\
 & & & & & & &
\end{array}\right)
\end{array}
$$

In the block column indexed by T', we chose a column ℓ' such that $\ell' \in L = [c+1, \dots, n_0 - k_0 + c - 1, t_1]$. We call ℓ the index of ℓ' in the list L.

We now consider a submatrix of $\boldsymbol{B}^{(\boldsymbol{\tau})}$ where we have chosen to keep the column ℓ' in each block column $T', T'_1, \dots, T'_{d+1}$. The only rows that are not zero on those columns are the rows ℓ in block rows indexed by T_1 to T_{d+1} as well as $S \cup \{c\}, S \subseteq T'$. We obtain the following matrix:

$$
\begin{array}{c}
\\
T_1 \\
T_i \\
T_j \\
T_{d+1} \\
\hline
T' \backslash \{t_i, t_j\} \cup \{c\}
\end{array}
\begin{array}{c}
\begin{array}{c|cccc}
T' & T'_1 & T'_i & T'_j & T'_{d+1}
\end{array} \\
\left(\begin{array}{c|cccc}
g_{\ell,t_1} & g_{\ell,c} & 0 & 0 & 0 \\
g_{\ell,t_i} & 0 & g_{\ell,c} & 0 & 0 \\
g_{\ell,t_j} & 0 & 0 & g_{\ell,c} & 0 \\
g_{\ell,t_{d+1}} & 0 & 0 & 0 & g_{\ell,c} \\
\hline
0 & 0 & g_{\ell,t_j} & g_{\ell,t_i} & 0
\end{array}\right)
\end{array}
$$

We name the columns of this matrix such that c_0 corresponds to the block column indexed by T' and c_k corresponds to the block columns indexed by T'_k for $k = 1, \dots, d+1$.

We want to cancel all g_{ℓ,t_i} in the first column c_0, we have to add the column c_i multiplied by $\frac{g_{\ell,t_i}}{g_{\ell,c}}$:

$$c_0 \leftarrow c_0 - \sum_{k=1}^{d+1} \frac{g_{\ell,t_k}}{g_{\ell,c}} \times c_k$$

On the rows indexed by $T_1, \dots, T_{d+1}$ we get zero on the column c_0. The operations that we do have no consequences on the row $S \cup \{c\}, S \subset T'$: we add twice the same coefficient and thus still have a zero afterwards.

Finally, in the first $\binom{c-1}{d+1}$ block columns (whose column indexing sets do not contain c), we have d columns that have at least one coefficient. In the remainder of the matrix, that is to say the $\binom{c-1}{d}$ columns shows indexing sets contains c, each block is made of a unique column of coefficients as well as a diagonal part of size $n_0 - k_0$ (with a permutation if needed). That means that there are $n_0 - k_0 + d$ nonzero columns.

It is to be noticed that if the computed rank is too high compared to the number of rows of the matrix, the rank is actually this number of rows. This

provides an expression for the rank :

$$\begin{aligned} Rank\left(\boldsymbol{B}^{(\tau)}\right) &= min\left\{\#Rows, \binom{c-1}{d+1}d + \binom{c-1}{d}(n_0-k_0+d)\right\} \\ &= min\left\{\binom{c}{d}(n_0-k_0), \binom{c}{d+1}d + \binom{c-1}{d}(n_0-k_0)\right\} \end{aligned}$$

□

Remark 6.7. In Theorem 6.6, we made 2 hypotheses about coefficients being different from 0.

If there is a zero coefficient in $\boldsymbol{B}_c$, it is possible to modify the order of the sets T and T' such that the matrices we use to cancel the others don't have any zero coefficients. This would not change the rank of the matrix. In the extreme case that all the B_s possible have a zero on the same row, i.e. it is impossible to find an order that works, this induces a decrease of the rank by one for each row full of zero. This can happen with a probability of $\frac{1}{((n_0-k_0)q)^c}$

If there is a column full of zero in $\boldsymbol{G}_f$, then there is one less column that has to be considered when counting the rank. This can happen with a probability of $\frac{1}{(cq)^{n_0-k_0}}$

In both cases, the probability is quite low. However, if the rank is actually smaller than expected, this is an advantage: we obtain more syzygies and thus more degree falls.

We have seen the two parts of the matrix $\boldsymbol{B}$ separately. Considering the two parts are independent and that there is no reason to have identical columns, we can simply add the two ranks we got before. We can deduce that the rank of the complete matrix is the following one:

Theorem 6.8. *The total rank of $\boldsymbol{B}$ is*

$$Rank(\boldsymbol{B}) \leq \min\left\{\binom{c}{d}(n_0-k_0), \binom{c-1}{d}(n_0-k_0) + \binom{c}{d+1}(d+\gamma-2)\right\} \tag{6.9}$$

6.3 Changing the Variables of DAGS Attack System by Minors

In this section, we apply the method explained in 3.3 on the system made from DAGS. We also explain what makes the process of creating the Macaulay matrix and calculating its rank different from the generic case.

6.3.1 Macaulay matrix

We begin by searching for the Macaulay matrix structure. We will see that it is similar though different from the generic case, as we add some new constraints linked to the codes used.

We have the following system:

$$\left(\sum_{i=1}^{k_0}\tau_i\begin{pmatrix}\mathbf{0} & \left(-\boldsymbol{G}_{\mathrm{f}}^T\right)_{[*,i]} & \mathbf{0}\end{pmatrix} + \sum_{i=1}^{n_0-k_0-1}\tau_{k_0+i}\begin{pmatrix}\mathbf{0}\\ \left(\boldsymbol{G}_{\mathrm{f}}^T\right)_{[i,*]}\\ \mathbf{0}\end{pmatrix} + \sum_{i=1}^{\gamma} b_i\left(\boldsymbol{H}_i\right)_{[*,\{1\dots k_0\}]}\right)\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \boldsymbol{U}^T\end{pmatrix} = \mathbf{0}_{(n_0-k_0)\times(k_0-c)}$$

We search for a different way to describe the system, in order to highlight the coefficients of the Macaulay matrices and the associated minors. We use the same process as in section 3.3, where we separated the minors of bidegree $(1,d)$ and $(1,d+1)$. This case is a bit harder due to the groups of variables that are not in every product with minors. It results in the following expression, for a given d:

$$\sum_{i=1}^{k_0}\left[\sum_{j\in J}\tau_i\boldsymbol{M}_{i,[\ell,j]}\det\left(\boldsymbol{U}_{[J\setminus\{j\},T]}\right)+\sum_{s\notin T}\tau_i\boldsymbol{M}_{i,[\ell,s]}\det\left(\boldsymbol{U}_{[J,T\cup\{s\}]}\right)\right]_{\ell=1,\dots,n}$$
$$+\sum_{i=k_0+1}^{n_0-1}\left[\sum_{j\in J}\tau_i\boldsymbol{M}_{i,[\ell,j]}\det\left(\boldsymbol{U}_{[J\setminus\{j\},T]}\right)+\sum_{s\notin T}\tau_i\boldsymbol{M}_{i,[\ell,s]}\det\left(\boldsymbol{U}_{[J,T\cup\{s\}]}\right)\right]_{\ell=1,\dots,n}$$
$$+\sum_{i=1}^{\gamma}\left[\sum_{j\in J}b_i\boldsymbol{M}_{n_0+i-1,[\ell,j]}\det\left(\boldsymbol{U}_{[J\setminus\{j\},T]}\right)+\sum_{s\notin T}b_i\boldsymbol{M}_{n_0+i-1,[\ell,s]}\det\left(\boldsymbol{U}_{[J,T\cup\{s\}]}\right)\right]_{\ell=1,\dots,n}$$

with the following matrices:

$$\boldsymbol{M}_i=\begin{cases}\begin{pmatrix}\mathbf{0} & \left(-\boldsymbol{G}_f^T\right)_{[*,\ell]} & \mathbf{0}\end{pmatrix} & \text{if } \ell=1,\dots,k_0\\ \begin{pmatrix}\mathbf{0}\\ \left(\boldsymbol{G}_f^T\right)_{[\ell-k_0,*]}\\ \mathbf{0}\end{pmatrix} & \text{if } \ell=k_0+1,\dots,n_0-1\\ \left(\boldsymbol{H}_{\ell-n_0+1}\right)_{[*,\{1,\dots,k_0\}]} & \text{if } \ell=n_0,\dots,n_0+\gamma-1\end{cases}$$

We obtain a Macaulay matrix with a form that is similar to the one of the generic case described in 3.3. However, we have additional constraints that comes from the sparsity of the matrices $\boldsymbol{M}_i$ that imply that the block matrices are going to be either sparser than before or completely filled by zeros. The constraints are dependent on each one of the three parts of the previously stated expression.

In the first part, that is about the variables τ_1 to τ_{k_0}, we can notice that $\boldsymbol{M}_{i,[\ell,j]}$ is zero unless $j=i$, due to the form of the matrix itself. The same thing applies for $\boldsymbol{M}_{i,[\ell,s]}$, which is zero unless $s=i$. In the second part, that is about the variables τ_{k_0+1} to τ_{n_0-1}, we see that $\boldsymbol{M}_{i,[\ell,j]}$ is zero unless $\ell=i$, due to the

matrix begin a null matrix apart from the ith column. The exact same thing goes for $\boldsymbol{M}_{i,[\ell,s]}$, which is zero unless $\ell = i$. The third part does not actually bring a new constraint, as the matrices $\boldsymbol{M}_i$ used in this part are not sparse.

Those constraints bring a lot more zeros to the Macaulay matrix than in the generic process. Indeed, the coefficients of the matrices $\boldsymbol{M}_i$ that are used directly correspond to coefficient of the matrix. Thus, they are inducing a lowering of the rank, and thus are modifying the complexity of solving the system. The difference between the generic rank and the DAGS' one is explained in next subsection.

6.3.2 Rank of the Macaulay matrix

We want to focus now on the rank of the Macaulay matrix we described just before.

While studying the generic case, we explained that the rank was:

$$\text{Rank}\left(\text{Macaulay matrix}\right) = \min\left\{\sum_{i=d_b}^{d_e} m\binom{\nu}{i+1}\binom{r}{i}, \left(\sum_{i=d_b}^{d_e} n\binom{\nu}{i}\binom{r}{i}\right) - 1\right\}$$

Unfortunately, due to the different structure of DAGS that we described in subsection 6.3.1, the rank is different. Considering that the number of zeros is higher in the Macaulay matrix for DAGS, it is reasonable to think that this rank is going to be lower. We remind the observation made in 3.3 regarding the fact that we can find the matrices $\boldsymbol{B}$ in the Macaulay matrix. We can extend it to the DAGS cryptosystems, by applying the same principle but with the rank of the matrix $\boldsymbol{B}$ for DAGS:

$$\begin{aligned}\text{Rank}\left(\text{Macaulay matrix}\right) = \min\Bigg\{&\sum_{i=d_b}^{d_e} (n_0 - k_0)\binom{k_0 - c}{i+1}\binom{c}{i},\\ &\sum_{i=d_b}^{d_e}\binom{k_0 - c}{i+1}\left(\binom{c-1}{i}(n_0 - k_0) + \binom{c}{i+1}(i + \gamma - 1)\right)\Bigg\}\end{aligned}$$

6.4 Experimental Results

We present here the experimental results of the application of the theoretical works previously explained on the DAGS cryptosystem.

6.4.1 With Input Improvements

We apply the improvements we just presented on the DAGS attack system, with the 3 sets of parameters. We have the choice of shortening the codes used or not. We will provide explanations and numerical results for both of those cases.

6.4.1.1 Solving without shortening

Let's first consider the case where we don't shorten the codes. The number of quadratic equations is $(k_0 - c) \times (n_0 - k_0 - 1)$ and the number of different variables are :

- $\boldsymbol{U}$: $N_{rows}(\boldsymbol{U}) \times c$, with the number of rows between 2 and $k_0 - c$.
- $\boldsymbol{T}$: $n_0 - k_0 + c - 1$
- $\boldsymbol{A}$: $\gamma - 2$

When we replace all these parameters by their value depending on the set of parameters of DAGS, we obtain the following table (we consider the maximum possible size for the matrix $\boldsymbol{U}$):

Table 6.1: Number of variables and equations for DAGS versions without shortening

Name	$N_{rows}(\boldsymbol{U})$	$\#U$	$\#T$	$\#A$	Vars	Eq.	Ratio
DAGS-1	22	88	29	2	119	550	4.6
DAGS-3	12	48	25	3	76	252	3.3
DAGS-5	9	18	23	4	45	189	4.2

These numbers are quite high, but we still have way more equations than variables, as indicated by the ratios. The choice to not apply a shortening to the system allows the degree reached during the Gröbner basis computation to stay the lowest possible, as we consider all the equations at our disposal. The drawback of this choice is that by taking into account all these equations, we slow down the computations: the computer has to test much more S-polynomials. The numerical results for DAGS are given in 6.4.1.3.

6.4.1.2 Solving with a shortening

Let's now consider the case where we chose to shorten the codes. We see the apparition of the variable a_0 in the formulas, which corresponds to the number of variables T_i that we remove. It also reduces the maximum possible size of the matrix $\boldsymbol{U}$.

The number of equations for this case is $(k_0 - c - a_0) \times (n_0 - k_0 - 1)$ and the number of variables are:

- $\boldsymbol{U}$: $N_{rows}(\boldsymbol{U}) \times c$, with the number of rows between 2 and $k_0 - c - a_0$.
- $\boldsymbol{T}$: $n_0 - k_0 + c - 1$
- $\boldsymbol{A}$: $\gamma - 2$

One thing that we can notice here is that the number of T_i variables that we obtain does not depend on a_0. We can do the following computation to find the number of T_i variables:

$$\begin{aligned} & (\#T_i - \#\text{shortened indices}) - \#\text{polynomial linear in } T_i - \#\text{specializations} \\ = \quad & (n_0 - a_0) - (k_0 - a_0 - c) - 1 \\ = \quad & n_0 - a_0 - k_0 + a_0 + c - 1 \\ = \quad & n_0 - k_0 + c - 1 \end{aligned}$$

As for the previous case where we were not shortening the code, we replace parameters by their values and obtain the following numbers of variables and equations (we consider the maximum possible size for $\boldsymbol{U}$ again):

Table 6.2: Number of variables and equations for each DAGS version with shortening

Name	$N_{rows}(\boldsymbol{U})$	$\#U$	$\#T$	$\#A$	Vars	Eq.
DAGS-1	$22 - a_0$	$88 - 4a_0$	29	2	$119 - 4a_0$	$550 - 26a_0$
DAGS-3	$12 - a_0$	$48 - 4a_0$	25	3	$76 - 4a_0$	$252 - 22a_0$
DAGS-5	$9 - a_0$	$18 - 2a_0$	23	4	$45 - 2a_0$	$189 - 22a_0$

With this table we clearly see that each time we increase a_0, we decrease the variables respectively by 4, 4 and 2 , while we remove respectively 26, 22 and 22 equations. It means that we can increase the ratio between equations and variables faster than we increase a_0. We can also notice that it entirely depends on the number of rows of the matrix $\boldsymbol{U}$. Here we can not provide a ratio as it depends on the value we chose for a_0, but the numerical results for different number of rows for $\boldsymbol{U}$ are presented in the next subsection.

6.4.1.3 Numerical results

We will present here the numerical results of our attacks on the different sets of parameters of DAGS. We will consider each set of parameters one after the other, as the behavior of the systems are not always the same. We attacked each set of parameters 100 times, and computed the means for the number of Gröbner bases computations, the duration of these computations and the memory used for them. We realized those tests using Magma [21] on a machine with a Intel® Xeon® 2.60GHz processor. The number of computations to obtain the Gröbner bases are indicated in the tables as the number of clock cycles of the CPU, given by Magma using the `ClockCycles()` function. Finally, the last column of the tables corresponds to the highest degree reached during the Gröbner basis computation. We can see on all the tables that the number of variables in $\#V$ does not vary, as we explained in subsection 6.4.1.2.

Results for DAGS_1 DAGS_1 is the first set of parameters, its security is supposed to be equivalent to the security of a 128 bits AES scheme. Here is the table summarizing what we found during our tests for different numbers of rows for $\boldsymbol{U}$:

Table 6.3: Numerical results of the attack on DAGS_1

$N_{rows}(\boldsymbol{U})$	$\#U$	$\#V$	Eq.	Ratio	Gröb.	Time	Mem.(Gb)	Deg.
2	8	31	50	1.28	2^{39}	276s	2.21	4
3	12	31	75	1.74	2^{38}	163s	1.11	4
4	16	31	100	2.13	2^{33}	4s	0.12	3
5	20	31	125	2.45	2^{34}	6s	0.24	3
22	88	31	550	4.6	2^{44}	108m	5.01	3

The first row of the table corresponds to the tests with the minimum number of variables $\boldsymbol{U}$, which are the same that were done in [16]. We are able to compute the Gröbner basis faster thanks to the improvements that we applied, 276 seconds instead of 19 minutes, but we can detect that the results are still not the best ones. This can be explained by observing the last column: when we keep only 2 or 3 rows for $\boldsymbol{U}$, the degree reached during the Gröbner basis algorithm is 4, which is higher than with more rows. It is due to the lower number of equations, the algorithm then needs to reach a higher degree to find the linear polynomials that we are searching for.

On the contrary, the last row shows the results for the tests without removing any row of the matrix $\boldsymbol{U}$. We notice that the duration of the attack is much higher, with 108 minutes, mainly because of the huge number of variables and equations: 119 variables for 550 equations. However, the degree reached during the Gröbner basis computation is only 3. This is a beneficial effect of having many equations, we have enough interactions between polynomials at a lower degree to find our basis.

We can notice that the best result is obtained for a matrix $\boldsymbol{U}$ with 4 rows. It corresponds to the lowest number of rows for $\boldsymbol{U}$ for which the degree stays 3. For these tests, we have 47 variables for 100 equations, which gives us a ratio of 2.13. It only takes 4 seconds and 120 Megabytes to compute the Gröbner basis.

Results for DAGS_3 DAGS_3 is the second set of parameters, its security is supposed to be equivalent to the security of a 192 bits AES scheme. Here is the table summarizing what we found during our tests for different numbers of rows for $\boldsymbol{U}$:

The first row of the table corresponds to the tests with the minimum number of variables $\boldsymbol{U}$, which is 8, 28 variables $\#V$ and 42 equations. This gives a ratio between variables and equations of 1.17. Our results are coherent with those from [16]: we were not able to perform the attack when we only keep 2 rows from $\boldsymbol{U}$. The computations had to stop because the memory on our server was saturated. We reached 139 Gigabytes and a degree 6 as long as the process was

Table 6.4: Numerical results of the attack on DAGS_3

$N_{rows}(\boldsymbol{U})$	$\#U$	$\#V$	Eq.	Ratio	Gröb.	Time	Mem.(Gb)	Deg.
2	8	28	42	1.17	–	–	≥ 139	≥ 6
3	12	28	63	1.58	2^{39}	321s	1.24	4
4	16	28	84	1.91	2^{37}	70s	1.11	4
5	20	28	105	2.19	2^{38}	140s	1.48	4
12	48	28	252	3.3	2^{44}	109m	10.2	4

running.

The last row shows the results for the tests without removing any row of the matrix $\boldsymbol{U}$. As for DAGS_1, the duration of the tests with the maximum of 12 rows was 109 minutes. Even if the ratio is high for those values, the number of variables and equations remains big so the computations are slower than with fewer rows. We can notice in the table that the degree reached for every case tested but the first is 4, and as the last row shows a degree of 4, the computations can not go lower.

We see that the best result is obtained for a matrix $\boldsymbol{U}$ with 4 rows, where the computation only takes about 70 seconds. The degree reached is the same than when we do not remove any row from $\boldsymbol{U}$, but we can explain the better results by the lower number of variables and equations, respectively 40 and 63 instead of 76 and 252.

Results for DAGS_5 DAGS_5 is the third and last set of parameters, its security is supposed to be equivalent to the security of a 256 bits AES scheme. Here is the table summarizing what we found during our tests for different numbers of rows for $\boldsymbol{U}$:

Table 6.5: Numerical results of the attack on DAGS_5

$N_{rows}(\boldsymbol{U})$	$\#U$	$\#V$	Eq.	Ratio	Gröb.	Time	Mem.(Gb)	Deg.
2	4	27	42	1.35	2^{31}	0.4s	0.12	3
3	6	27	63	1.91	2^{31}	0.4s	0.13	3
4	8	27	84	2.40	2^{31}	0.5s	0.15	3
5	10	27	105	2.84	2^{31}	0.4s	0.19	3
9	18	27	189	4.2	2^{33}	42s	0.30	3

The case of DAGS_5 is simpler than the two others, as the results do not change a lot depending on the number of rows of $\boldsymbol{U}$ we are keeping. As a general rule for this set of parameter, the less variables and equations there are, the shortest the Gröbner basis computation is. Thus, we can notice that we obtain the best results for a matrix $\boldsymbol{U}$ with only 2 rows. The degree reached during the computations is 3 and stays the same for all the tests we did. The duration of those computations as well as the memory used are very small: 42

seconds and 300 Megabytes for the full matrix $\boldsymbol{U}$ with 9 rows, and less than 0.5 seconds and 200 Megabytes with 2 to 5 rows.

6.4.2 With the $\boldsymbol{B}$ matrix

We now present the results that are induced by the method explained before. We compare them to the solving of the DAGS system without any precomputation.

Table 6.6: Comparison of the computation of a Gröbner basis for DAGS systems with different sets of parameters with or without precomputations, for the best degree found in 6.4.1.3

Params	Precomp. time	GB time	Total time	Without Precomp.
DAGS_1	0.3s	1.1s	1.4s	4s
DAGS_3	0.3s	1.6s	1.9s	70s
DAGS_5	0.0s	0.3s	0.3s	0.4s

Let us analyse the results for each set of parameters, as the results are slightly different for each of them.

For DAGS_1, the time is divided by almost 3. We can notice that this is a slight reduction of the time complexity. The difference between the methods with or without precomputations is pretty low for one computation, although it can become significative when we multiply the systems to solve.

For DAGS_3, the time is divided approximately by 35. This is the set of parameters which shows the best improvement thanks to the precomputation. It was the hardest set to attack without the precomputations, and although it stays that way with this method, the difference with the other sets has decreased immensely.

For DAGS_5, the time does not seem to change a lot. This can be explained by the fact that this set of parameters is already fast without any precomputations. We can chose any of the two methods as they imply a similar time complexity.

6.4.3 With MinRank Attack

We present the experimental results that we obtained for the MinRank Attack.

Table 6.7: Experimental results of rank, sizes of matrices and time complexity for each set of parameters. Comparison with previous results.

Params	Matrix Rank	Matrix Size	Time Comp.	Time With Precomp.
DAGS_1	1226	1248×2448	0.1s	1.4s
DAGS_3	2485	2640×4250	0.9s	1.9s
DAGS_5	191	198×310	0.01s	0.3s

In Table 6.7, we can see that the matrix sizes are really small, and that the rank is smaller than the number of rows, which means that we have a decrease in the rank. These two columns directly impacted the time taken by the algorithm to find a Gröbner Basis. The Time Comp. columns corresponds to the time taken to do an Echelon Form on the Macaulay Matrix, while the Time With Precomp. corresponds to the time taken to compute the F4 algorithm in MAGMA with some precomputations. By comparing those two columns, we clearly see the improvements. For all sets of parameters tested, the time complexity drops under the second, which is a really good time for an attack.

6.5 Attacking New Parameters

After the attack presented in [16], the authors of DAGS updated their parameters to get a better security and thus be protected from it. We already explained what were the main changes in 5.3.4, here in table 6.8 we simply remind these updated parameters.

Table 6.8: DAGS updated sets of parameters

	q	m	n	k	γ	t	NIST Security Level
DAGS_1.1	2^6	2	832	416	4	13	Level 1 (128 bits AES)
DAGS_3.1	2^8	2	1216	512	5	11	Level 3 (192 bits AES)
DAGS_5.1	2^8	2	1600	896	5	11	Level 5 (256 bits AES)

With these new parameters, the attack as we presented it previously does not work because the parameters are too big. We will give the details about this first, then we will introduce the concept of a hybrid method, following by the explanation on how it allows us to attack the DAGS_1.1 set of parameters. We will end this section by giving the reasons why we can not apply the same method to the two other sets of parameters.

6.5.1 Using the previous attack on updated parameters

To try to attack these updated parameters as we did with the first sets, we need to calculate how many variables and equations we have:

Table 6.9: Number of variables and equations for the updated parameters

Param.	$N_{rows}(\boldsymbol{U})$	c	$\#U$	$\#V$	Var.	Eq.	Ratio
DAGS_1.1	18	8	144	35	179	450	2.5
DAGS_3.1	0	16	–	–	–	0	0
DAGS_3.1 dual	6	16	96	34	130	90	0.7
DAGS_5.1	12	16	192	40	232	252	1.1

These numbers do not allow us to perform the attack as we did before. The reasons are different for each set of parameters:

DAGS_1.1 The number of variables that we have for this set of parameters is too high for the computations to end without exceeding the memory we had. Reducing the number of rows of $\boldsymbol{U}$ diminishes the number of variables accordingly but it does not help with the complexity. The computation involve a matrix that has more than 2 millions of rows with polynomials in degree 4, which was too much to be handled. We were not able to attack this set of parameters using the previous attack, but the ratio being 2.5, we tried to find an other way: we explain what we managed to do in 6.5.2.

DAGS_3.1 For this set of parameters, we have $c = k_0$. It implies that some matrices can not have any rows, and thus the code $\mathscr{D}$ does not exist and we can not perform the attack at all. We even thought of considering the dual of the public code, but then the ratio between variables and equations is 0.7, which makes the system underdetermined and impossible to solve.

DAGS_5.1 The problem with this set of parameters is similar to the one encountered for DAGS_1.1. Indeed, the numbers of variables and equations, respectively 232 and 252, is too high to allow us to attack the system. The ratio being as low as 1.1, it may be difficult to apply the same reasoning as we did with DAGS_1.1 to attack it another way.

6.5.2 Hybrid method

Before getting into how we can apply it to DAGS_1.1, we will begin by giving the general idea about what an hybrid method is in our case.

6.5.2.1 General Principle

The principle of this method was introduced in [20]: it consists in mixing exhaustive search to the Gröbner bases computations we were already using before.

Using only one of these 2 methods is too complex: we know that the complexity of computing Gröbner bases for the updated parameters is too high, and that the complexity for exhaustive search corresponds to the size of the field of the variables raised to the number of variables we have in the system, which is too big. Moreover, using this technique increases the ratio between equations and variables, which is, as we have seen in 6.4.1.3, is beneficial for the complexity of the attack. Our goal by using this method is to find the best tradeoff that allow us to compensate the gain obtained by working on systems with less variables with the cost of the exhaustive search performed on some given variables.

Furthermore, trying to compute a Gröbner basis for an incorrect specialization of variables is slightly faster, as it stops as soon as it finds 1 in the system. It means that the system we tested has no solution, and we then know that the specialization was wrong. We repeat that for each specializations, taking

advantage of the fact that we do less operations when the algorithm does not have a solution.

We remind that the system we are trying to attack is bilinear, which means that when we specialize some variables chosen carefully, the polynomials containing those same variables can become linear. This directly impacts the complexity of the attack, as linear polynomials are easier to deal with.

6.5.2.2 Application on DAGS

To apply this on DAGS, we need to select which ones of the variables are the best ones to specialize: we have the choice between the variables $U \in \mathbb{F}_q$, $A \in \mathbb{F}_{q^m}$ and $T \in \mathbb{F}_{q^m}$. We need to take into account the size of the space they are contained in, which affect the complexity of the exhaustive search, as well as the number of equations that are using those variables. Indeed, as the polynomials are bilinear, specializing variables means making them linear if we chose the variables correctly.

If we look at how the variables are distributed among the blocks of polynomials, we notice that one block uses all variables A and T, and only one row of variables U. It means that if we specialize some variables A or T, we keep all the polynomials bilinear, whereas specializing a row of variables U allows us to get a complete block composed of solely linear variables. Moreover, as we said, the variables U are in a finite field smaller than the one the variables A and T are, so it will cost less to enumerate all their possible values.

Through our tests and experimentations, we noticed that it is more effective to specialize complete rows of variables U at the same time. Indeed, only applying the exhaustive search on a limited part of the row can help by reducing the number of variables but it is not enough to make the polynomials linear.

The complexity of computing a Gröbner basis while using an exhaustive search is:

$$q^{N_{spec}} \times C_{false} + C_{true} \tag{6.10}$$

where N_{spec} corresponds to the number of variables we are trying to specialize, and C_{true} and C_{false} are respectively the complexities of the Gröbner basis computation when the specialization of the variables is right and wrong.

6.5.3 Results on the updated parameters

Now that we have seen the principle of this method and the general idea about how we want to apply it on the updated DAGS sets of parameters, let us see the results we obtained. We will see that this method works on the set of parameters DAGS_1.1, and we will explain why it does not on the other sets of parameters.

6.5.3.1 Results on DAGS_1.1

We begin by considering the set of parameters DAGS_1.1. We remind that the parameters as well as the initial numbers of variables and equations for DAGS_1.1 can be found in the tables 6.8 and 6.9.

We saw in the previous subsection that the variables we need to chose to be the most efficient in our attack are the variables U. For this set of parameters, we have $c = 8$ variables U on each row. Let us begin by trying to specialize only one row, and testing with different number of rows for the matrix $\boldsymbol{U}$. We obtain the following results, the last column not being experimental but corresponding to the application of the formula 6.10:

Table 6.10: Number of variables, equations and experimental complexities for the hybrid method applied to DAGS_1.1. C_{false} and C_{true} respectively correspond to the complexity of a try that does not work and the complexity of a try that works.

$N_{rows}(\boldsymbol{U})$	Var	Linear	Bilinear	C_{false}	C_{true}	Total
2	43	25	25	2^{35}	2^{36}	2^{83}
3	51	25	50	2^{35}	2^{36}	2^{83}
4	59	25	75	2^{38}	2^{39}	2^{86}
5	67	25	100	2^{40}	2^{40}	2^{88}

We can see on this table of experimental complexities, that for all those tests, the total is far under the claimed security of 128 bits for DAGS_1.1. It might be too big to be computed practically for the moment, but the fact that these values are less than 2^{128} are sufficient to consider this hybrid method a success for this set of parameters. We can also notice in this table that the number of linear equations is completely related to the number of rows that we are specializing in $\boldsymbol{U}$. Indeed, 1 row specialized gives us a linear block. However, the number of bilinear equations grows at the same time as the number of rows of $\boldsymbol{U}$.

We now have a successful attack against this updated set of parameters, and we can wonder if we can improve it further, for example by specializing more variables. We can try to do the calculations : if we want to try to specialize more variables, we need to consider two rows of the matrix $\boldsymbol{U}$. That corresponds to 16 variables. If we apply the formula of complexity 6.10, we have:

$$\left(2^6\right)^{16} \times C_{false} + C_{true} = 2^{96} \times C_{false} + C_{true}$$

The value 2^{96} is below the claimed security of 128 bits, but it is not clear that the product with the complexity of a Gröbner basis computation with a wrong specialization will not exceed 2^{128}. In that case, that attack would not be considered successful. The best choice for the number of rows for $\boldsymbol{U}$ to get the best results in terms of complexity seems to be 2 or 3, with the best measured complexity of 2^{83}. Let us now see if we can apply the same reasoning to the 2 other sets of parameters.

6.5.3.2 On the impossibility of attacking DAGS_3.1 and DAGS_5.1

We can not use the same attack on the two other sets of parameters. We will tackle them one after the other as the reasons are completely different.

For DAGS_3.1, we actually already explained in 6.5.1 why this attack can not work. The parameters of this set are preventing the code $\mathscr{D}$ to exist, thus making the attack impossible. The dual is not a viable solution either, due to the ratio between equations and variables being under 1.

For DAGS_5.1, the reason is in the complexity of the attack using the hybrid method. Indeed, to be efficient, we need to specialize at least 1 complete row of variables U, which corresponds to 16 variables here. We made the table 6.11 for different number of rows of $\boldsymbol{U}$ and only 1 row specialized.

Table 6.11: Number of variables and equations and ratio for the hybrid method on DAGS_5.1

$N_{rows}(\boldsymbol{U})$	Var	Linear	Bilinear	Ratio
2	72	21	21	0.75
3	88	21	42	0.88
4	104	21	63	0.95
5	120	21	84	1.01
6	136	21	105	1.05

We can see in this table that the ratios are all too small and the number of variables too high to be able to attack the set of parameters DAGS_5.1. As specializing 16 variables seems not enough, we can consider specializing more variables.

Let us consider the case where we want to specialize 2 rows of the matrix $\boldsymbol{U}$, which corresponds to 32 variables. We can apply the formula 6.10 and we obtain:

$$\left(2^8\right)^{32} \times C_{false} + C_{true} = 2^{256} \times C_{false} + C_{true}$$

As stated previously, C_{false} corresponds to the complexity of a specialization that does not work, whereas C_{true} corresponds to the complexity for a correct specialization. It is clear here that it is not relevant to consider the case where we try to specialize 32 variables as the cost of the exhaustive search alone is already 2^{256}, and it corresponds to the claimed security for this set of parameters. We would have to add the C_{false} and C_{true} into the formula, which will inevitably surpass the claimed security. We can conclude by saying that with all those elements, we are unable to attack DAGS_5.1 with the hybrid method.

6.5.4 Another update on the parameters

Following this hybrid method for attacking DAGS, the authors published in [4] another update on the sets of parameters, which consisted in raising the size

of the finite field for the DAGS_1.1 set, as well as reducing the values for the parameters n, k and t:

Table 6.12: DAGS last sets of parameters

	q	m	n	k	γ	t	NIST Security Level
DAGS_1.1	2^8	2	704	352	4	11	Level 1 (128 bits AES)
DAGS_3.1	2^8	2	1216	512	5	11	Level 3 (192 bits AES)
DAGS_5.1	2^8	2	1600	896	5	11	Level 5 (256 bits AES)

With these changes, we can legitimately wonder if these parameters are now secure against this hybrid method for attacking. There were no changes for DAGS_3.1 and DAGS_5.1, so we can already rule them out. For DAGS_1.1, we need to do some calculations. The codimension c value changes from 8 to $\frac{m \times q}{2^\gamma} = \frac{2 \times 2^8}{2^4} = 2^5 = 32$, which means that a row of the matrix $\boldsymbol{U}$ corresponds now to 32 variables. Using the formula 6.10, we determine that the complexity of computing a Gröbner basis while specializing 32 variables is:

$$\left(2^8\right)^{32} \times C_{false} + C_{true} = 2^{256} \times C_{false} + C_{true}$$

We remind that C_{false} is the complexity of a try for a wrong specialization, and C_{true} is the complexity for a correct specialization. As for DAGS_5.1, we can see that the complexity of the exhaustive search alone is too big for the calculations to be made. These changes on the set of parameters DAGS_1.1 are protecting the system against this hybrid version of the attack we presented in this chapter.

Conclusion

In this thesis we first explained what are polynomial systems and one method to solve them: Gröbner bases computation. We then detailed how we can adapt it to some subfamilies, in particular the bilinear polynomial systems. We focused on bilinear systems that can be expressed as a product of matrices, as some new results were published during these last years. We studied those, and tried to make some changes to get a better understanding of the process as well as improving the complexity. We managed to do so by using the minors as variables of the systems, making the Macaulay matrices much smaller. This consequently decreased the complexity of applying linear algebra on the matrix.

Following this analysis that was done on generic cases, we recalled definitions that are important in code-based cryptography. We discussed how the process of the first part can be applied to the solving of the system coming from on attack on the DAGS cryptosystem. After presenting how is the DAGS scheme working, we detailed the first attack that was done against it. Everything we explained after that were improvements that we made on this attack by modify the modeling. We improved it first by altering the inputs of the system, making the attack successful on a set of parameters that was not broken yet. The other modifications we tried in order to further improve the attack were more focused on the Gröbner Basis process, particularly by pre-computing some equations with a lower complexity than it would have been with the Gröbner basis computation algorithm.

Bibliography

[1] "Algorithms for matrix canonical forms". PhD thesis. 2000. DOI: 10.3929/ETHZ-A-004141007 (page 19).

[2] Josh Alman and Virginia Vassilevska Williams. "A Refined Laser Method and Faster Matrix Multiplication". In: *CoRR* abs/2010.05846 (2020). arXiv: 2010.05846. URL: https://arxiv.org/abs/2010.05846 (page 19).

[3] John B. Baena, Daniel Cabarcas, and Javier A. Verbel. "On the Complexity of Solving Generic Over-determined Bilinear Systems". In: *CoRR* abs/2006.09442 (2020). URL: https://arxiv.org/abs/2006.09442 (pages 3, 31, 41).

[4] Gustavo Banegas, Paulo S.L.M. Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiecoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti, and Jefferson E. Ricardini. "DAGS: Reloaded Revisiting Dyadic Key Encapsulation". In: *Code-Based Cryptography - 7th International Workshop, CBC 2019, Darmstadt, Germany, May 18-19, 2019, Revised Selected Papers*. Ed. by Marco Baldi, Edoardo Persichetti, and Paolo Santini. Vol. 11666. Lecture Notes in Computer Science. Springer, 2019, pp. 69–85. DOI: 10.1007/978-3-030-25922-8_4 (page 110).

[5] Gustavo Banegas, Paulo S.L.M Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiecoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti, and Jefferson E. Ricardini. *DAGS : Key Encapsulation for Dyadic GS Codes*. Specifications from the first round submission to the NIST post-quantum cryptography call - Version 1. 2017. URL: https://www.dags-project.org/pdf/DAGS_spec_v1.pdf (pages 71, 73).

[6] Gustavo Banegas, Paulo S.L.M Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiecoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti, and Jefferson E. Ricardini. *DAGS : Key Encapsulation for Dyadic GS Codes*. Specifications from the first round submission to the NIST post-quantum cryptography call - Version 2. 2018. URL:

https://www.dags-project.org/pdf/DAGS_spec_v2.pdf (pages 71, 73).

[7] Gustavo Banegas, Paulo Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiecoumba Gueye, Richard Haeussler, Jean Klamti, Ousmane Ndiaye, Duc Tri Nguyen, Edoardo Persichetti, and Jefferson E. Ricardini. "DAGS: Key encapsulation using dyadic GS codes". In: *Journal of Mathematical Cryptology* 12.4 (2018), pp. 221–239. DOI: 10.1515/jmc-2018-0027 (page 71).

[8] Magali Bardet. "Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie". PhD thesis. Université Pierre et Marie Curie - Paris VI, 2004. URL: https://tel.archives-ouvertes.fr/tel-00449609 (page 21).

[9] Magali Bardet and Manon Bertin. "Improvement of algebraic attacks for solving superdetermined MinRank instances". In: *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, September 28-30, 2022, Proceedings*. Ed. by Jung Hee Cheon and Thomas Johansson. Vol. 13512. Lecture Notes in Computer Science. Springer, 2022 (pages 3, 47).

[10] Magali Bardet, Manon Bertin, Alain Couvreur, and Ayoub Otmani. "Practical Algebraic Attack on DAGS". In: *CBC 2019*. Darmstadt, Germany, May 2019. ISBN: 978-3-030-25921-1 (pages 3, 87).

[11] 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". In: *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*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. Lecture Notes in Computer Science. Springer, 2020, pp. 64–93. URL: https://doi.org/10.1007/978-3-030-45727-3%5C_3 (page 31).

[12] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier A. Verbel. "Improvements of Algebraic Attacks for Solving the Rank Decoding and MinRank Problems". In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 507–536. URL: https://doi.org/10.1007/978-3-030-64837-4%5C_17 (pages 3, 31, 34, 47, 48, 51).

[13] Magali Bardet, Jean Charles Faugère, Bruno Salvy, and Bo Yin Yang. "Asymptotic Behaviour of the Index of Regularity of Quadratic Semi-Regular Polynomial Systems". In: *MEGA*. Porto Conte, Alghero, Sardinia, Italy, 2005, p. 16 (page 29).

[14] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. "On the Complexity of the F5 Gröbner basis Algorithm". In: *CoRR* abs/1312.1655 (2013). arXiv: 1312.1655. URL: http://arxiv.org/abs/1312.1655 (page 20).

[15] Elise Barelli. "On the security of short McEliece keys from algebraic and algebraic geometry codes with automorphisms. (Étude de la sécurité de certaines clés compactes pour le schéma de McEliece utilisant des codes géométriques)". PhD thesis. University of Paris-Saclay, France, 2018. URL: https://tel.archives-ouvertes.fr/tel-01993634 (pages 76, 81).

[16] Élise Barelli and Alain Couvreur. "An Efficient Structural Attack on NIST Submission DAGS". In: *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*. Ed. by Thomas Peyrin and Steven D. Galbraith. Vol. 11272. Lecture Notes in Computer Science. Springer, 2018, pp. 93–118. DOI: 10.1007/978-3-030-03326-2_4 (pages 3, 70, 74, 75, 77, 78, 83, 84, 87, 103, 106).

[17] Elwyn R Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968. ISBN: 9780070049031 (page 63).

[18] Elwyn R Berlekamp. "Goppa Codes". In: *IEEE Trans. Inf. Theory* 19 (1973), pp. 590–592 (page 62).

[19] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagenand Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. *Classic McEliece: conservative code-based cryptography*. Specifications from the first round submission to the NIST post-quantum cryptography call. 2017. URL: https://classic.mceliece.org/nist/mceliece-20201010.pdf (page 68).

[20] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. "Hybrid approach for solving multivariate systems over finite fields". In: *Journal of Mathematical Cryptology* 3.3 (2009), pp. 177–197 (page 107).

[21] Wieb Bosma, John Cannon, and Catherine Playoust. "The Magma Algebra System I: The User Language". In: *J. Symb. Comput.* 24.3/4 (1997), pp. 235–265. DOI: 10.1006/jsco.1996.0125. URL: https://doi.org/10.1006/jsco.1996.0125 (page 102).

[22] B. Buchberger. "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms". In: *SIGSAM Bull.* 10.3 (1976), pp. 19–29. DOI: 10.1145/1088216.1088219 (page 17).

[23] Jonathan F. Buss, Gudmund S. Frandsen, and Jeffrey O. Shallit. "The Computational Complexity of Some Problems of Linear Algebra". In: 58.3 (1999), pp. 572–596 (page 30).

[24] D. Cox, J. Little, and D. OSHEA. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer New York, 2013. ISBN: 9781475726930 (pages 11, 18, 21, 28).

[25] D.A. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer New York, 2005. ISBN: 9780387207063 (page 11).

[26] P. Delsarte. "On Subfield Subcodes of Modified Reed-Solomon Codes". In: *IEEE Trans. Inf. Theor.* 21.5 (1975), pp. 575–576. ISSN: 0018-9448. DOI: 10.1109/TIT.1975.1055435 (pages 60, 61).

[27] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638 (page 56).

[28] David Eisenbud. "Commutative Algebra: with a View Toward Algebraic Geometry". In: vol. 150. Texts in Mathematics. Springer-Verlag, 1995 (page 21).

[29] Jean-Charles Faugère. "A new efficient algorithm for computing Gröbner bases (F4)". In: *Journal of Pure and Applied Algebra* 139.1-3 (June 1999), pp. 61–88. DOI: 10.1016/S0022-4049(99)00005-5. URL: https://hal.archives-ouvertes.fr/hal-01148855 (page 17).

[30] Jean-Charles Faugère. "A new efficient algorithm for computing Gröbner bases without reduction to zero F5". In: *International Symposium on Symbolic and Algebraic Computation Symposium - ISSAC 2002*. Colloque avec actes et comité de lecture. internationale. Villeneuve d'Ascq, France: ACM, July 2002, pp. 75–83. DOI: 10.1145/780506.780516. URL: https://hal.inria.fr/inria-00100995 (page 17).

[31] Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. "Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1, 1): Algorithms and complexity". In: *J. Symb. Comput.* 46.4 (2011), pp. 406–437. URL: https://doi.org/10.1016/j.jsc.2010.10.014 (pages 29, 37).

[32] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. "Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering". In: *J. Symb. Comput.* 16.4 (1993), pp. 329–344. DOI: https://doi.org/10.1006/jsco.1993.1051 (page 15).

[33] Ernst Gabidulin. "Theory of codes with maximum rank distance (translation)". In: *Problems of Information Transmission* 21 (Jan. 1985), pp. 1–12 (page 30).

[34] V. D. Goppa. "A New Class of Linear Error-correcting Codes". In: *Problems of Info. Transmission* 6.3 (1970), pp. 207–212. ISSN: 0555-2923 (page 62).

[35] Hermann J. Helgert. "Noncyclic Generalizations of BCH and Srivastava Codes". In: *Inf. Control.* 21.3 (1972), pp. 280–290. DOI: 10.1016/S0019-9958(72)80007-X (page 63).

[36] Hermann J. Helgert. "Srivastava codes". In: *IEEE Trans. Inf. Theory* 18.2 (1972), pp. 292–297. DOI: 10.1109/TIT.1972.1054760 (page 63).

[37] Hermann J. Helgert. "Alternant Codes". In: *Inf. Control.* 26.4 (1974), pp. 369–380. DOI: 10.1016/S0019-9958(74)80005-7 (page 61).

[38] Hermann J. Helgert. "Binary Primitive Alternant Codes". In: *Inf. Control.* 27.2 (1975), pp. 101–108. DOI: 10.1016/S0019-9958(75)90099-6 (page 61).

[39] D. Hilbert. *Ueber die Theorie der algebraischen Formen*. B.G. Teubner, 1890 (page 18).

[40] D. Hilbert. *Theory of Algebraic Invariants*. Trans. by S. Marxsen, D. Hilbert, B. Sturmfels, R.C. Laubenbacher, R. Laubenbacher, and H. David. Cambridge Mathematical Library. Cambridge University Press, 1993. ISBN: 9780521449038 (page 18).

[41] W.V.D. Hodge and D. Pedoe. *Methods of Algebraic Geometry: Volume 2*. Cambridge Mathematical Library. Cambridge University Press, 1994. ISBN: 9780521469012 (page 47).

[42] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. "A Modular Analysis of the Fujisaki-Okamoto Transformation". In: *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. Lecture Notes in Computer Science. Springer, 2017, pp. 341–371. DOI: 10.1007/978-3-319-70500-2_12 (page 72).

[43] Claude-Pierre Jeannerod, Clément Pernet, and Arne Storjohann. "Rank-profile revealing Gaussian elimination and the CUP matrix decomposition". In: *CoRR* abs/1112.5717 (2011). URL: http://arxiv.org/abs/1112.5717 (page 19).

[44] Aviad Kipnis and Adi Shamir. "Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization". In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 19–30. URL: https://doi.org/10.1007/3-540-48405-1%5C_2 (page 30).

[45] Daniel Lazard. "Gröbner-Bases, Gaussian elimination and resolution of systems of algebraic equations." In: *EUROCAL*. Ed. by J. A. van Hulzen. Vol. 162. Lecture Notes in Computer Science. Springer, 1983, pp. 146–156 (pages 17, 19, 21).

[46] F. S. Macaulay. "Some Formulæ in Elimination". In: *Proceedings of the London Mathematical Society* s1-35.1 (1902), pp. 3–27. DOI: https://doi.org/10.1112/plms/s1-35.1.3 (pages 17, 20).

[47] F. S. Macaulay. *The Algebraic Theory of Modular Systems*. Cambridge Mathematical Library. Cambridge University Press, 1916. ISBN: 9780521455626 (page 20).

[48] F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes*. Vol. 16. North-Holland Mathematical Library, 1977 (pages 57, 63, 64, 72, 73).

[49] Robert J. McEliece. "A Public-Key System Based on Algebraic Coding Theory". In: DSN Progress Report 44. Jet Propulsion Lab, 1978, pp. 114–116 (pages 66, 67).

[50] Harald Niederreiter. "Knapsack-type cryptosystems and algebraic coding theory". In: *Problems of Control and Information Theory* 15.2 (1986), pp. 157–166 (page 67).

[51] NIST. *Post-Quantum Cryptography Call for Proposals*. 2017. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals (pages 66, 68).

[52] Ryo Nojima, Hideki Imai, Kazukuni Kobara, and Kirill Morozov. "Semantic security for the McEliece cryptosystem without random oracles". In: *Des. Codes Cryptogr.* 49.1-3 (2008), pp. 289–305. DOI: 10.1007/s10623-008-9175-9 (page 71).

[53] N. Patterson. "The algebraic decoding of Goppa codes". In: *IEEE Transactions on Information Theory* 21.2 (1975), pp. 203–207. DOI: 10.1109/TIT.1975.1055350 (page 62).

[54] I. S. Reed and G. Solomon. "Polynomial Codes Over Certain Finite Fields". In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304. DOI: 10.1137/0108018 (page 60).

[55] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342 (page 56).

[56] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700 (page 56).

[57] Javier A. Verbel, John Baena, Daniel Cabarcas, Ray A. Perlner, and Daniel Smith-Tone. "On the Complexity of "Superdetermined" Minrank Instances". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*. Ed. by Jintai Ding and Rainer Steinwandt. Vol. 11505. Lecture Notes in Computer Science. Springer, 2019, pp. 167–186. URL: https://doi.org/10.1007/978-3-030-25510-7%5C_10 (pages 3, 31, 34, 35, 37, 38, 40, 95).

[58] Bruns W. and Vetter U. *Determinantal Rings*. Vol. 1327. Lecture Notes in Computer Science. Springer, 1988 (page 47).

Appendix A

Products and Vectorization

We present here the products that are used a few times in this thesis, as well as vectorization and their respective properties.

A.1 Different Products in this Thesis

A.1.1 Classical product

Definition A.1. *The* classical *product of two matrices* $\boldsymbol{A}$ *of size* $n_A \times m_A$ *and* $\boldsymbol{B}$ *of size* $n_B \times m_B$ *is defined as:*

$$(\boldsymbol{A} \cdot \boldsymbol{B})_{i,j} = (\boldsymbol{AB})_{i,j} = \sum_{k=1}^{m_A} a_{i,k} b_{k,j}$$

It works only if $m_A = n_B$ *and the resulting matrix is* $n_A \times m_B$*.*

A.1.2 Kronecker product $\otimes$

Definition A.2. *The Kronecker product of two matrices* $\boldsymbol{A}$ *of size* $n_A \times m_A$ *and* $\boldsymbol{B}$ *of size* $n_B \times m_B$ *corresponds to the multiplication of each coefficient of the matrix* A *by the whole matrix* $\boldsymbol{B}$*. It can be defined as:*

$$(\boldsymbol{A} \otimes \boldsymbol{B})_{i,j} = a_{(i-1) \text{ div } n_B+1,(j-1) \text{ div } m_B+1} b_{(i-1) \bmod n_B+1,(j-1) \bmod m_B+1}$$

The resulting matrix is $n_A n_B \times m_A m_B$*.*

Example A.3.

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1m_A}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{n_A1}\boldsymbol{B} & \cdots & a_{n_Am_A}\boldsymbol{B} \end{bmatrix}$$

A.1.3 Star product $\star$

We remind here the definition of the star product we used earlier in this thesis:

Definition A.4. *The star product of two matrices $\boldsymbol{A}$ of size $n_A \times m_A$ and $\boldsymbol{B}$ of size $n_B \times m_B$ corresponds to the component-wise product of every combination of one row from each matrix. It is defined as:*

$$(\boldsymbol{A} \star \boldsymbol{B})_{i,j} = a_{(i-1) \text{ div } n_B+1,j} b_{(i-1) \text{ mod } n_B+1,j}$$

It works only if $m_A = m_B$ and the resulting matrix is $n_A n_B \times m_A$. For this specific product, an order has to be chosen for the rows that are taken. The choice here is taking one row on the first matrix, and multiplying it by every row of the second matrix.

Example A.5.

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1m_A}b_{1m_A} \\ a_{11}b_{21} & a_{12}b_{22} & \cdots & a_{1m_A}b_{2m_A} \\ \vdots & \vdots & & \vdots \\ a_{11}b_{n_B1} & a_{12}b_{n_B2} & \cdots & a_{1m_A}b_{n_Bm_A} \\ a_{21}b_{11} & a_{22}b_{12} & \cdots & a_{2m_A}b_{1m_A} \\ \vdots & \vdots & & \vdots \\ a_{n_A1}b_{n_B1} & a_{n_A2}b_{n_B2} & \cdots & a_{n_Am_A}b_{n_Bm_A} \end{bmatrix}$$

A.2 Properties of the Products

Lemma A.6. *Let $\boldsymbol{v}$ a vector of size $m_{\boldsymbol{v}}$ and $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two matrices with $m_{\boldsymbol{v}}$ columns.*

$$(\boldsymbol{X} \star \boldsymbol{v}) \cdot \boldsymbol{Y}^T = \boldsymbol{X} \cdot (\boldsymbol{Y} \star \boldsymbol{v})^T \tag{A.1}$$

$$(\boldsymbol{X} \star \boldsymbol{Y})\, \boldsymbol{v}^T = \text{vec}_{row}\left((\boldsymbol{X} \star \boldsymbol{v})\, \boldsymbol{Y}^T\right) \tag{A.2}$$

Proof. **Equation A.1.** Using the definitions of the products that are used, we can directly prove the equality :

$$\begin{aligned} \left(\boldsymbol{X}\,(\boldsymbol{Y} \star \boldsymbol{v})^t\right)_{i,j} &= \sum_{k=1}^{m_X} \boldsymbol{X}_{i,k}\,(\boldsymbol{Y} \star \boldsymbol{v})^t_{k,j} \\ &= \sum_{k=1}^{m_X} \boldsymbol{X}_{i,k}\,(\boldsymbol{Y} \star \boldsymbol{v})_{j,k} \\ &= \sum_{k=1}^{m_X} \boldsymbol{X}_{i,k} \boldsymbol{Y}_{(j-1) \text{ div } 1+1,k} \boldsymbol{v}_{(j-1) \text{ mod } 1+1,k} \\ &= \sum_{k=1}^{m_X} \boldsymbol{X}_{i,k} \boldsymbol{Y}_{j,k} \boldsymbol{v}_{1,k} \end{aligned}$$

$$= \left((\boldsymbol{X} \star \boldsymbol{v}) . \boldsymbol{Y}^t\right)_{i,j}$$

Equation A.2. We begin by computing the formula for the coefficient of $(\boldsymbol{X} \star \boldsymbol{Y}) . \boldsymbol{v}^t$. The result is a vector, si we only need one indexing variable :

$$\begin{aligned}\left((\boldsymbol{X} \star \boldsymbol{Y})\, \boldsymbol{v}^t\right)_i &= \sum_{k=1}^{m_V} (\boldsymbol{X} \star \boldsymbol{Y})_{i,k}\, \boldsymbol{v}_k \\ &= \sum_{k=1}^{m_V} \boldsymbol{X}_{(i-1) \text{ div } n_Y+1,k} \boldsymbol{Y}_{(i-1) \bmod n_Y+1,k} \boldsymbol{v}_k\end{aligned}$$

We look at the second part. First, we compute the coefficient of $(X \star V) . Y^t$:

$$\begin{aligned}\left((\boldsymbol{X} \star \boldsymbol{v})\, \boldsymbol{Y}^t\right)_{i,j} &= \sum_{k=1}^{m_Y} (\boldsymbol{X} \star \boldsymbol{v})_{i,k} \boldsymbol{Y}^t_{k,j} \\ &= \sum_{k=1}^{m_Y} \boldsymbol{X}_{(i-1) \text{ div } 1+1,k} \boldsymbol{v}_{(i-1) \bmod 1+1,k} \boldsymbol{Y}_{j,k} \\ &= \sum_{k=1}^{m_Y} \boldsymbol{X}_{i,k} \boldsymbol{v}_{1,k} \boldsymbol{Y}_{j,k}\end{aligned}$$

We can finish the proof by applying the definition of the row vectorization:

$$\begin{aligned}\text{vec}_{row}\left((\boldsymbol{X} \star \boldsymbol{v})\, \boldsymbol{Y}^t\right)_i &= \left((\boldsymbol{X} \star \boldsymbol{v})\, \boldsymbol{Y}^t\right)_{(i-1) \text{ div } n_Y+1,(i-1) \bmod n_Y+1} \\ &= \sum_{k=1}^{m_Y} \boldsymbol{X}_{(i-1) \text{ div } n_Y+1,k} \boldsymbol{v}_{1,k} \boldsymbol{Y}_{(i-1) \bmod n_Y+1,k} \\ &= \left((\boldsymbol{X} \star \boldsymbol{Y})\, \boldsymbol{v}^t\right)_i\end{aligned}$$

□

Lemma A.7. *Let $\boldsymbol{A}$ and $\boldsymbol{C}$ two matrices with the same number of columns, and $\boldsymbol{b}$ and $\boldsymbol{d}$ two vectors of the same size.*

$$(\boldsymbol{A} \otimes \boldsymbol{b}) \star (\boldsymbol{C} \otimes \boldsymbol{d}) = (\boldsymbol{A} \star \boldsymbol{C}) \otimes (\boldsymbol{b} \star \boldsymbol{d}) \tag{A.3}$$

Proof. To prove this equality, we compute the formula for a coefficient in each matrix. We note n_X and m_X respectively the number of rows and the number of columns of a matrix $\boldsymbol{X}$.

$$\begin{aligned}\Big((\boldsymbol{A} \otimes \boldsymbol{b}) \star (\boldsymbol{C} \otimes \boldsymbol{d})\Big)_{i,j} &= (\boldsymbol{A} \otimes \boldsymbol{b})_{(i-1) \text{ div } n_C+1,j} \cdot (\boldsymbol{C} \otimes \boldsymbol{d})_{(i-1) \bmod n_C+1,j} \\ &= \boldsymbol{A}_{(i-1) \text{ div } n_C+1,(j-1) \text{ div } m_B+1} \boldsymbol{b}_{1,(j-1) \bmod m_B+1}\end{aligned}$$

$$\begin{aligned}
& \cdot \boldsymbol{C}_{(i-1) \bmod n_C+1,(j-1) \operatorname{div} m_D+1} \boldsymbol{d}_{1,(j-1) \bmod m_D+1} \\
\Big((\boldsymbol{A} \star \boldsymbol{C}) \otimes (\boldsymbol{b} \star \boldsymbol{d})\Big)_{i,j} &= (\boldsymbol{A} \star \boldsymbol{C})_{i,(j-1) \operatorname{div} m_B+1} \cdot (\boldsymbol{b} \star \boldsymbol{d})_{1,(j-1) \bmod m_D+1} \\
&= \boldsymbol{A}_{(i-1) \operatorname{div} n_C+1,(j-1) \operatorname{div} m_B+1} \\
& \cdot \boldsymbol{C}_{(i-1) \bmod n_C+1,(j-1) \operatorname{div} m_D+1} \\
& \cdot \boldsymbol{b}_{1,(j-1) \bmod m_B+1} \boldsymbol{d}_{1,(j-1) \bmod m_D+1}
\end{aligned}$$

We can see that we obtain the same formula for the coefficients of each matrix, thus our matrices are equivalent. □

A.3 Vectorization: Definitions and Properties

Lemma A.8. *For all matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, we have :*

$$\operatorname{vec}_{row}(\boldsymbol{ABC}) = \left(\boldsymbol{A} \otimes \boldsymbol{C}^T\right) \operatorname{vec}_{row}(\boldsymbol{B}) \tag{A.4}$$

Proof. We will try to express each side of the equality and prove that they are equal.

Left Part. Using the definitions of products given in Section A.1, we can write the coefficient (i, j) of the product $\boldsymbol{ABC}$ as:

$$(\boldsymbol{ABC})_{i,j} = \sum_{k=1}^{m_{AB}} (\boldsymbol{AB})_{i,k} \boldsymbol{C}_{k,j} = \sum_{k'=1}^{m_B} \left(\sum_{k''=1}^{m_A} \boldsymbol{A}_{i,k''} \boldsymbol{B}_{k'',k'} \right) \boldsymbol{C}_{k',j}$$

The matrix we obtain is of size $n_A \times m_C$ with the following equalities between the dimensions : $m_A = n_B$ and $m_B = n_C$. We can now use the definition of row vectorization to obtain :

$$\begin{aligned}
(vec_{row}(\boldsymbol{ABC}))_i &= (\boldsymbol{ABC})_{(i-1) \operatorname{div} m_C+1,(i-1) \bmod m_C+1} \\
&= \sum_{k'=1}^{m_B} \left(\sum_{k''=1}^{m_A} \boldsymbol{A}_{(i-1) \operatorname{div} m_C+1,k''} \boldsymbol{B}_{k'',k'} \right) \boldsymbol{C}_{k',(i-1) \bmod m_C+1}
\end{aligned}$$

The final vector has a size of $n_A m_C$.

Right part. Using the definitions of the products in Section A.1 again, we have :

$$\begin{aligned}
\left(\boldsymbol{A} \otimes \boldsymbol{C}^t\right)_{i,j} &= \boldsymbol{A}_{(i-1) \operatorname{div} n_{(C^t)}+1,(j-1) \operatorname{div} m_{(C^t)}+1} \boldsymbol{C}^t_{(i-1) \bmod n_{(C^t)}+1,(j-1) \bmod m_{(C^t)}+1} \\
&= \boldsymbol{A}_{(i-1) \operatorname{div} m_C+1,(j-1) \operatorname{div} n_C+1} \boldsymbol{C}^t_{(i-1) \bmod m_C+1,(j-1) \bmod n_C+1} \\
&= \boldsymbol{A}_{(i-1) \operatorname{div} m_C+1,(j-1) \operatorname{div} n_C+1} \boldsymbol{C}_{(j-1) \bmod n_C+1,(i-1) \bmod m_C+1}
\end{aligned}$$

We obtain a matrix of size $n_A m_C \times m_A n_C$. Using the row vectorization we can write :

$$(vec_{row}(\boldsymbol{B}))_i = \boldsymbol{B}_{(i-1) \operatorname{div} m_B+1,(i-1) \bmod m_B+1}$$

We obtain a vector of size $n_B m_B$ that we need to multiply with $vec_{row}(\boldsymbol{ABC})$:

$$\left((\boldsymbol{A}\otimes\boldsymbol{C}^t)vec_{row}(\boldsymbol{B})\right)_i = \sum_{k=1}^{m_A n_C} \left(\boldsymbol{A}\otimes\boldsymbol{C}^t\right)_{i,k} \left(vec_{row}(\boldsymbol{B})\right)_k$$

$$= \sum_{k=1}^{m_A n_C} \boldsymbol{A}_{(i-1) \text{ div } m_C+1,(k-1) \text{ div } n_C+1}\boldsymbol{C}_{(k-1) \bmod n_C+1,(i-1) \bmod m_C+1}$$

$$\cdot \boldsymbol{B}_{(k-1) \text{ div } m_B+1,(k-1) \bmod m_B+1}$$

As we want to prove that both parts of the lemma 3.5 are equal, we can use the constraint : $m_B = n_C$

$$= \sum_{k=1}^{m_A m_B} \boldsymbol{A}_{(i-1) \text{ div } m_C+1,(k-1) \text{ div } m_B+1}\boldsymbol{C}_{(k-1) \bmod m_B+1,(i-1) \bmod m_C+1}$$

$$\cdot \boldsymbol{B}_{(k-1) \text{ div } m_B+1,(k-1) \bmod m_B+1}$$

Let k be written as $k = (k'-1) + (k''-1)m_B + 1$, with $k' = 1 \dots m_B$ and $k'' = 1 \dots m_A$

$$= \sum_{k'=1}^{m_B}\sum_{k''=1}^{m_A} \boldsymbol{A}_{(i-1) \text{ div } m_C+1,k''}\boldsymbol{C}_{k',(i-1) \bmod m_C+1}\boldsymbol{B}_{k'',k'}$$

$$= \sum_{k'=1}^{m_B}\left(\sum_{k''=1}^{m_A} \boldsymbol{A}_{(i-1) \text{ div } m_C+1,k''}\boldsymbol{B}_{k'',k'}\right)\boldsymbol{C}_{k',(i-1) \bmod m_C+1}$$

We obtain the same expression found in the left part. □

Appendix B

Canceling More Than the Homogeneous Part

Contents

During our research, we also tried to cancel a bigger part of the system that just the homogeneous part as we were previously doing.

B.1 Canceling More than the Homogeneous Part - Theory

Previously, we only worked on the homogeneous part of the algebraic system. We will give some variants by considering different parts of it. We remind this system:

$$\mathcal{F} : \boldsymbol{x}\boldsymbol{M}\boldsymbol{U}^T + \boldsymbol{x}\boldsymbol{A} + \boldsymbol{B}\boldsymbol{U}^T + \boldsymbol{C} = 0$$

We will try to apply the same method as in subsection 6.2.4 but we will modify it to find syzygies not only for the homogeneous part, but also for a piece of the affine part. For each possibility, we will explain the part that we are trying to cancel and the associated theoretical results.

B.1.1 Canceling all monomials with x

Our first variant that we present here consists in focusing on all the parts that are including x as an unknown. The idea behind doing so is that we want to keep the same modeling that we introduced before, while trying to cancel more monomials than the ones in the homogeneous part of highest degree, with the final goal to be able to do a complete linearization. It gives us the following set of polynomials that we want to cancel:

$$\boldsymbol{x}\boldsymbol{M}\boldsymbol{U}^T + \boldsymbol{x}\boldsymbol{A}$$

The products $\boldsymbol{x}\boldsymbol{M}\boldsymbol{U}^T$ and $\boldsymbol{x}\boldsymbol{A}$ can be rewritten as sums of products to obtain the following writing:

$$\begin{aligned}\boldsymbol{x}\boldsymbol{M}\boldsymbol{U}^T + \boldsymbol{x}\boldsymbol{A} &= \sum_{\ell=1}^{n} x_\ell \boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \sum_{\ell=1}^{n} x_\ell \boldsymbol{A}^{(\ell)} \\ &= \sum_{\ell=1}^{n} x_\ell \left(\boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{A}^{(\ell)}\right)\end{aligned}$$

We will now use the same method we explained in subsection 6.2.4, thus we begin by computing the jacobian matrix containing the partial derivatives in x.

$$\begin{aligned}&\mathrm{jac}_{\boldsymbol{x}}\left(\mathrm{vec}_{row}\left(\sum_{\ell=1}^{n} x_\ell \left(\boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{A}^{(\ell)}\right)\right)\right) \\ &= \left(\mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{A}^{(\ell)}\right)\right)_{\ell=1,\dots,n} \\ &= \left(\mathrm{vec}_{row}\left(\begin{pmatrix}\boldsymbol{M}^{(\ell)} & \boldsymbol{A}^{(\ell)}\end{pmatrix}\begin{pmatrix}\boldsymbol{U} & \boldsymbol{I}_\nu\end{pmatrix}^T\right)\right)_{\ell=1,\dots,n} \\ &= \left(\boldsymbol{I}_m \otimes \begin{pmatrix}\boldsymbol{U} & \boldsymbol{I}_\nu\end{pmatrix}\right)\left(\mathrm{vec}_{row}\left(\begin{pmatrix}\boldsymbol{M}^{(\ell)} & \boldsymbol{A}^{(\ell)}\end{pmatrix}\right)\right)_{\ell=1,\dots,n}\end{aligned}$$

Let us recall the set J and define the new T:

$$\begin{aligned}&J = (j_1, \dots, j_{d+1}) && \text{such that } 1 \leq j_k < j_{k+1} \leq \nu \\ &T = (t_1, \dots, t_d) && \text{such that } 1 \leq t_k < t_{k+1} \leq r + \nu\end{aligned}$$

By analogy with the previous section, we can write the definition of the vector $\boldsymbol{v}_{J,T}$. We remind that all the analysis is done for characteristic 2 so the signs are not considered. The process works a similar way for a characteristic greater than 2, the only difference is the possibility of having minus signs.

$$\left(\boldsymbol{v}_{J,T}\right)_j = \begin{cases}\det\left(\begin{pmatrix}\boldsymbol{U} & \boldsymbol{I}_\nu\end{pmatrix}_{[J\setminus\{j\},T]}\right) & \text{if } j \in J \\ 0 & \text{else}\end{cases}$$

We can actually be more precise about this vector. Indeed, depending on the set of indices T, we can have different values and there are more cases where the value is zero. We split T into two parts :

$$T_1 \subset \{1, \dots, r\} \qquad\qquad T_1 = T \cap \{1, \dots, r\}$$

$$T_2 \subset \{1,\dots,\nu\} \qquad T_2 = T \cap \{r+1,\dots,r+\nu\} - r$$
$$T \subset \{1,\dots,r+\nu\} \qquad T = T_1 \cup (T_2 + r)$$

Thanks to these new sets, we can define the vector $\boldsymbol{v}_{J,T}$ only with $\boldsymbol{U}$:

$$(\boldsymbol{v}_{J,T})_j = \begin{cases} \det\left(\boldsymbol{U}_{[J\setminus\{j\}\setminus T_2,T_1]}\right) & \text{if } j \in J \text{ and } T_2 \subseteq J\setminus\{j\} \\ 0 & \text{else} \end{cases}$$

We notice that the value if the determinant can be 1 if T_1 is empty.

We can then deduce the following product:

$$\begin{aligned}
&\boldsymbol{v}_{J,T} \cdot \begin{pmatrix} \boldsymbol{U} & \boldsymbol{I}_\nu \end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{v}_{J,T}\boldsymbol{U} & \boldsymbol{v}_{J,T}\boldsymbol{I}_\nu \end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{v}_{J,T}\boldsymbol{U} & \boldsymbol{v}_{J,T} \end{pmatrix} \\
&= \sum_{\substack{s\in\{1,\dots,r\} \\ s\notin T_1}} \det\left(\boldsymbol{U}_{[J\setminus T_2,T_1\cup\{s\}]}\right) e_s^{r+\nu} + \sum_{\substack{s\in\{1,\dots,\nu\} \\ s\ \notin T_2}} \det\left(\boldsymbol{U}_{[J\setminus\{s\}\setminus T_2,T_1]}\right) e_{r+s}^{r+\nu}
\end{aligned}$$

We keep the same definition for $\boldsymbol{V}_J$: $\boldsymbol{V}_J = \sum_T \boldsymbol{a}_T \otimes \boldsymbol{v}_{J,T}$

Similarly to what we did in the proof of Theorem 3.8, we search for elements in the kernel of the jacobian by searching for elements in the kernel of the matrix B_J. We multiply the vector of minors with the jacobian matrix in the x variables of the part we aim to cancel:

$$\begin{aligned}
&\boldsymbol{V}_J \cdot \mathrm{jac}_{\boldsymbol{x}} \left(\mathrm{vec}_{row} \left(\sum_{\ell=1}^{n} x_\ell \left(\boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{A}^{(\ell)} \right) \right) \right) \\
&= \sum_T \left(\boldsymbol{a}_T \otimes \boldsymbol{v}_{J,T}\right) \left(\boldsymbol{I}_m \otimes \begin{pmatrix} \boldsymbol{U} & \boldsymbol{I}_\nu \end{pmatrix}\right) \left(\mathrm{vec}_{row}\left(\begin{pmatrix} \boldsymbol{M}^{(\ell)} & \boldsymbol{A}^{(\ell)} \end{pmatrix}\right)\right)_{\ell=1,\dots,n} \\
&= \sum_T \left(\boldsymbol{a}_T\boldsymbol{I}_m \otimes \boldsymbol{v}_{J,T} \begin{pmatrix} \boldsymbol{U} & \boldsymbol{I}_\nu \end{pmatrix}\right) \left(\mathrm{vec}_{row}\left(\begin{pmatrix} \boldsymbol{M}^{(\ell)} & \boldsymbol{A}^{(\ell)} \end{pmatrix}\right)\right)_{\ell=1,\dots,n} \\
&= \sum_T \left(\boldsymbol{a}_T \otimes \left(\sum_{\substack{s\in\{1,\dots,r\} \\ s\notin T_1}} \det\left(\boldsymbol{U}_{[J\setminus T_2,T_1\cup\{s\}]}\right) e_s^{r+\nu} \right.\right. \\
&\quad \left.\left. + \sum_{\substack{s\in\{1,\dots,\nu\} \\ s\ \notin T_2}} \det\left(\boldsymbol{U}_{[J\setminus\{s\}\setminus T_2,T_1]}\right) e_{r+s}^{r+\nu} \right) \right) \times \left(\mathrm{vec}_{row}\left(\begin{pmatrix} \boldsymbol{M}^{(\ell)} & \boldsymbol{A}^{(\ell)} \end{pmatrix}\right)\right)_{\ell=1,\dots,n}
\end{aligned}$$

Through these computations, we see that our resulting expression can be divided in two parts: the first one is identical to the computations that were done in Theorem 3.8, when we consider that $J\setminus T_2$ is replacing J and T_1 is replacing T. Thus, we will focus on the computations on the second part here.

$$\sum_T \sum_{\substack{s\in\{1,\dots,\nu\}\\ s\ \notin T_2}} \det\left(\boldsymbol{U}_{[J\setminus\{s\}\setminus T_2,T_1]}\right)\left(\boldsymbol{a}_T\otimes e_{r+s}^{r+\nu}\right)\times\left(\mathrm{vec}_{row}\left(\begin{pmatrix}\boldsymbol{M}^{(\ell)} & \boldsymbol{A}^{(\ell)}\end{pmatrix}\right)\right)_{\ell=1,\dots,n}$$

$$=\sum_T \sum_{\substack{s\in\{1,\dots,\nu\}\\ s\ \notin T_2}} \det\left(\boldsymbol{U}_{[J\setminus\{s\}\setminus T_2,T_1]}\right)\boldsymbol{a}_T\left(\begin{pmatrix}\boldsymbol{M}^{(\ell)} & \boldsymbol{A}^{(\ell)}\end{pmatrix}_{[*,r+s]}\right)_{\ell=1,\dots,n}$$

$$=\sum_T \sum_{\substack{s\in\{1,\dots,\nu\}\\ s\ \notin T_2}} \det\left(\boldsymbol{U}_{[J\setminus\{s\}\setminus T_2,T_1]}\right)\boldsymbol{a}_T\left(\boldsymbol{A}^{(\ell)}_{[*,s]}\right)_{\ell=1,\dots,n}$$

This second part actually corresponds to the constraints we are adding to the system by trying to cancel more parts of it.

From these computations, we can deduce the matrix $\boldsymbol{B}_{T,T'}$ by following the formula given in the previous section:

$$\boldsymbol{B}_{T,T'}=\begin{cases}\left(\boldsymbol{M}^{(\ell)}_{[*,s]}\right)_{\ell=1,\dots,n} & \text{if } T'\setminus T=\{s\} \text{ and } s\in\{1,\dots,r\}\\ \left(\boldsymbol{A}^{(\ell)}_{[*,s-r]}\right)_{\ell=1,\dots,n} & \text{if } T'\setminus T=\{s\} \text{ and } s\in\{r+1,\dots,r+\nu\}\\ \boldsymbol{0}_{m\ \times n} & \text{else}\end{cases}$$

We can apply the corollary 3.10 to have the corresponding number of syzygies :

$$\binom{r+\nu}{d}m-\binom{r+\nu}{d+1}n$$

Now that we have the matrix that allow us to find elements from the kernel of the jacobian matrix and thus indice degree falls, we want to compute the effects of the product of these kernel elements with the affine part of the system. This means we can precompute the polynomials that are composing the system after the degree fall. To do so, we multiply the part of the system that was not canceled with the syzygies.

$$\begin{aligned}V_J\cdot\mathrm{vec}_{row}\left(\boldsymbol{b}\boldsymbol{U}^T+\boldsymbol{c}\right)&=\left(\sum_T \boldsymbol{a}_T\otimes\boldsymbol{v}_{J,T}\right)\cdot\mathrm{vec}_{row}\left(\boldsymbol{b}\boldsymbol{U}^T+\boldsymbol{c}\right)\\&=\sum_T\left(\boldsymbol{a}_T\otimes\boldsymbol{v}_{J,T}\right)\cdot\mathrm{vec}_{row}\left(\boldsymbol{b}\boldsymbol{U}^T+\boldsymbol{c}\right)\end{aligned}$$

We can use Equation (3.5):

$$=\sum_T \mathrm{vec}_{row}\left(\boldsymbol{a}_T\left(\boldsymbol{b}\boldsymbol{U}^T+\boldsymbol{c}\right)\boldsymbol{v}_{J,T}^T\right)$$

By definition, we can see that the order of the minors in the vector $\boldsymbol{v}_{J,T}$ depends on the size of the $\#T_1$ part. Thus, if we consider all J and all T, the total number of minors is : $\sum_{i=0}^{d}\binom{\nu}{i}\binom{r+\nu}{i}$.

From the product of $\boldsymbol{b}\boldsymbol{U}^T$ with $\boldsymbol{v}_{J,T}^T$, we have minors of higher order, but the principle is the same. Thus, the number of minors is : $\sum_{i=1}^{d+1} \binom{\nu}{i}\binom{r+\nu}{i}$ for all J and T.

We can notice that we cannot just add the two numbers we found here: some minors are the same. If we consider all T and all J, we have the same minors for orders from 1 to d. The total number of minors is then:

$$\sum_{i=0}^{d+1} \binom{\nu}{i}\binom{r+\nu}{i}$$

We now have the number of equations, thanks to the number of syzygies, as well as the numbers of minors, that we consider as our variables. We removed all x variables from our system, and we have removed the corresponding equations. The equations we have left after all those computations are of a lower degree, and only in minors of $\boldsymbol{U}$. Moreover, we consider those minors as our variables, making those remaining equations linear.

Thanks to the number of equations and variables, we can give an expression of the degree d_{min}, as we explained in Subsection 3.2.2.3:

$$d_{min} = \min\left\{d \mid \left[\binom{r+\nu}{d}m > \binom{r+\nu}{d+1}n\right], 1 \leq d \leq r+\nu-1\right\} \tag{B.1}$$

We remind that this d_{min} can be used to estimate the complexity if the computations end shortly after the first degree fall.

B.1.2 Other variants

We can try to cancel other parts of the system. We can either try to cancel all monomials with u variables, or try to cancel everything but the constant part of the system.

B.1.2.1 Canceling all monomials with $\boldsymbol{U}$

The second variant we present here consists in focusing on all the parts that are including variables from $\boldsymbol{U}$. It means that we want to cancel:

$$\sum_{\ell=1}^{n} x_\ell \boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{b}\boldsymbol{U}^T$$

We homogenize the system by adding a x_0 variable that is equal to 1 and considering that $\boldsymbol{M}^{(0)} = \boldsymbol{b}$. This allow to extend the sum to the index 0 and the computations are similar to those described originally in Section 3.2. The degree d_{min} used to compute the complexity estimation then becomes:

$$d_{min} = \min\left\{d \mid \left[\binom{r}{d}m > \binom{r}{d+1}(n+1)\right], 1 \leq d \leq r-1\right\} \tag{B.2}$$

B.1.2.2 Canceling all but the constant part

We can go even further than previously and merge the cancellation of all monomials containing x variables with the homogenization with the variable x_0. This means we have only the constant part remaining to be multiplied by the syzygies found during the computations.

The part we want to cancel for this last variant can then be written as:

$$\begin{aligned}\sum_{\ell=0}^{n} x_\ell \boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{x}\boldsymbol{A} &= \sum_{\ell=0}^{n} x_\ell \boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \sum_{\ell=1}^{n} x_\ell \boldsymbol{A}^{(\ell)} \\ &= \sum_{\ell=0}^{n} x_\ell \left(\boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{A}^{(\ell)}\right) \qquad \text{with } \boldsymbol{A}^{(0)} = \boldsymbol{0}\end{aligned}$$

As we merge both cancellation processes that we described previously, the complexity estimation for this variant is also similar. Indeed, the value of d_{min} is:

$$d_{min} = \min\left\{d \mid \left[\binom{r+\nu}{d} m > \binom{r+\nu}{d+1}(n+1)\right], 1 \leq d \leq r+\nu-1\right\} \quad \text{(B.3)}$$

However, the two variants presented in Subsection B.1.2.1 and B.1.2.2 do not improve computations enough to be used. Their complexity being similar to the one presented in B.1.1, we will chose to focus on this first variant, canceling monomials with x variables, to try to improve the attack on the DAGS cryptosystem.

B.2 Canceling More Than the Homogeneous Part of the DAGS System

Following the methods that we presented in Chapter 3, we want to extend the previous results by trying to cancel more than the homogeneous part of highest degree of the system.

B.2.1 Identification of the matrices

We remind the homogeneous part of highest degree and the affine part :

$$\begin{aligned}\mathcal{F}_h = \Bigg(&\sum_{i=1}^{c} \tau_{k_0-c+i} \begin{pmatrix} \boldsymbol{0} & \left(-\boldsymbol{G}_2^T\right)_{[*,i]} & \boldsymbol{0} \end{pmatrix} + \sum_{i=1}^{n_0-k_0-1} \tau_{k_0+i} \begin{pmatrix} \boldsymbol{0} \\ \left(\boldsymbol{G}_2^T\right)_{[i,*]} \\ \boldsymbol{0} \end{pmatrix} \\ &+ \sum_{i=3}^{\gamma} b_i \left(\boldsymbol{H}_i\right)_{[*,\{k_0-c+1\ldots k_0\}]} \Bigg) \cdot \boldsymbol{U}^T\end{aligned}$$

$$\mathcal{F}_a = \sum_{i=1}^{k_0-c} \tau_i \begin{pmatrix} \mathbf{0} & \left(-\boldsymbol{G}_1^T\right)_{[*,i]} & \mathbf{0} \end{pmatrix} + \sum_{i=1}^{n_0-k_0-1} \tau_{k_0+i} \begin{pmatrix} \mathbf{0} \\ \left(\boldsymbol{G}_1^T\right)_{[i,*]} \\ \mathbf{0} \end{pmatrix}$$
$$+ \sum_{i=3}^{\gamma} b_i \left(\boldsymbol{H}_i\right)_{[*,\{1\ldots k_0-c\}]} + \left(\boldsymbol{H}_2\right)_{[*,\{k_0-c+1\ldots k_0\}]} \cdot \boldsymbol{U}^T + \left(\boldsymbol{H}_2\right)_{[*,\{1\ldots k_0-c\}]}$$

What we need to do for each variant is to identify the matrix of the theoretical analysis to the matrices of DAGS. The identification is the following:

$$\boldsymbol{x}_\ell = \begin{cases} \tau_\ell & \text{if } \ell = 1, \ldots, n_0 - 1 \\ b_{\ell-n_0+1} & \text{if } \ell = n_0, \ldots, n_0 + (\gamma - 2) - 1 \end{cases}$$

$$\boldsymbol{M}^{(\ell)} = \begin{cases} \mathbf{0}_{(n_0-k_0)\times c} & \text{if } \ell = 1, \ldots, k_0 - c \\ \begin{pmatrix} \mathbf{0} & \left(-\boldsymbol{G}_2^T\right)_{[*,\ell-k_0+c]} & \mathbf{0} \end{pmatrix} & \text{if } \ell = k_0 - c + 1, \ldots, k_0 \\ \begin{pmatrix} \mathbf{0} \\ \left(\boldsymbol{G}_2^T\right)_{[\ell-k_0,*]} \\ \mathbf{0} \end{pmatrix} & \text{if } \ell = k_0 + 1, \ldots, n_0 - 1 \\ \left(\boldsymbol{H}_{\ell-n_0+1}\right)_{[*,\{k_0-c+1\ldots k_0\}]} & \text{if } \ell = n_0, \ldots, n_0 + (\gamma - 2) - 1 \end{cases}$$

$$\boldsymbol{A}^{(\ell)} = \begin{cases} \begin{pmatrix} \mathbf{0} & \left(-\boldsymbol{G}_1^T\right)_{[*,\ell]} & \mathbf{0} \end{pmatrix} & \text{if } \ell = 1, \ldots, k_0 - c \\ \mathbf{0}_{(n_0-k_0)\times(k_0-c)} & \text{if } \ell = k_0 - c + 1, \ldots, k_0 \\ \begin{pmatrix} \mathbf{0} \\ \left(\boldsymbol{G}_1^T\right)_{[\ell-k_0,*]} \\ \mathbf{0} \end{pmatrix} & \text{if } \ell = k_0 + 1, \ldots, n_0 - 1 \\ \left(\boldsymbol{H}_{\ell-n_0+1}\right)_{[*,\{1\ldots k_0-c\}]} & \text{if } \ell = n_0, \ldots, n_0 + (\gamma - 2) - 1 \end{cases}$$

$$\boldsymbol{B} = \left(\boldsymbol{H}_2\right)_{[*,\{k_0-c+1\ldots k_0\}]}$$
$$\boldsymbol{C} = \left(\boldsymbol{H}_2\right)_{[*,\{1\ldots k_0-c\}]}$$

We will begin with explaining how we can cancel all parts with τ and b.

B.2.2 Canceling all parts with τ and b variables

This corresponds to the method of subsection B.1.1 applied on DAGS. We want to cancel all parts with τ and b variables, that means we want to cancel $\mathcal{F}_h$ and the affine part that is linear in the τ and b variables:

$$\sum_{i=1}^{k_0-c} \tau_i \begin{pmatrix} \mathbf{0} & \left(-\boldsymbol{G}_1^T\right)_{[*,i]} & \mathbf{0} \end{pmatrix} + \sum_{i=1}^{n_0-k_0-1} \tau_{k_0+i} \begin{pmatrix} \mathbf{0} \\ \left(\boldsymbol{G}_1^T\right)_{[i,*]} \\ \mathbf{0} \end{pmatrix} + \sum_{i=3}^{\gamma} b_i \left(\boldsymbol{H}_i\right)_{[*,\{1\ldots k_0-c\}]}$$

Using the identifications made in the subsection B.2.1, we directly know the form of the jacobian matrix. However, to simplify the writing we will consider different cases corresponding to different values of ℓ.

$$\mathrm{jac}_{\tau,b}\left(\mathcal{F}_h + \mathcal{F}_a(\tau,b)\right) = \mathrm{vec}_{row}\left(\boldsymbol{M}^{(\ell)}\boldsymbol{U}^T + \boldsymbol{A}^{(\ell)}\right)_{\ell=1,\ldots,n_0+(\gamma-2)-1}$$

with $\mathcal{F}_a\left(\tau,b\right)$ the section of the affine part of the system that is linear in the τ and b variables.

We can apply the same process we did in B.1.1 and compute the equations that remain after canceling the homogeneous part of highest degree as well as the part that is linear in τ and b.

$$\begin{aligned}
&\boldsymbol{V}_J \cdot \mathrm{vec}_{row}\left(\left(\boldsymbol{H}_2\right)_{[*,\{k_0-c+1\ldots k_0\}]}\boldsymbol{U}^T + \left(\boldsymbol{H}_2\right)_{[*,\{1\ldots k_0-c\}]}\right)\\
&= \left(\sum_T \boldsymbol{a}_T \otimes \boldsymbol{v}_{J,T}\right) \cdot \mathrm{vec}_{row}\left(\left(\boldsymbol{H}_2\right)_{[*,\{k_0-c+1\ldots k_0\}]}\boldsymbol{U}^T + \left(\boldsymbol{H}_2\right)_{[*,\{1\ldots k_0-c\}]}\right)\\
&= \sum_T \left(\boldsymbol{a}_T \otimes \boldsymbol{v}_{J,T}\right) \cdot \mathrm{vec}_{row}\left(\left(\boldsymbol{H}_2\right)_{[*,\{k_0-c+1\ldots k_0\}]}\boldsymbol{U}^T + \left(\boldsymbol{H}_2\right)_{[*,\{1\ldots k_0-c\}]}\right)\\
&= \sum_T \mathrm{vec}_{row}\left(\boldsymbol{a}_T\left(\left(\boldsymbol{H}_2\right)_{[*,\{k_0-c+1\ldots k_0\}]}\boldsymbol{U}^T + \left(\boldsymbol{H}_2\right)_{[*,\{1\ldots k_0-c\}]}\right)\boldsymbol{v}_{J,T}^T\right)
\end{aligned}$$

This gives us the following number of syzygies and variables:

$$\begin{aligned}
N_{\text{syzygies}} &= \binom{k_0}{d}(n_0-k_0) - \binom{k_0}{d+1}(n_0-k_0+c-1)\\
N_{\text{monomials}} &= \sum_{i=0}^{d+1}\binom{k_0-c}{i}\binom{k_0}{i}
\end{aligned}$$

Using the number of syzygies, we can adapt the expression for the degree d_{min}, that helps with estimating the complexity of solving the system when the computations end shortly after the first degree fall. The degree d_{min} becomes:

$$d_{min} = \min\left\{d \mid \left[\binom{k_0}{d}(n_0-k_0) > \binom{k_0}{d+1}(n_0-k_0+c-1)\right], 1 \le d \le k_0-1\right\} \tag{B.4}$$

It can be then used in the complexity expression given in 3.2.2.3.

We are particularly interested in the practical complexity of solving the system created from the DAGS cryptosystem, that is to say the computations time, thus we present next the experimental results using the variant we described.

B.2.3 Experimental Results

We present the experimental results that we got with these modeling.

The other potential experiments where we cancel all u or everything but the constant part did not bring any improvements. For some of them, the method did not even succeed in finding syzygies for the system.

Table B.1: Experimental results for the different sets of parameters of DAGS, depending on the variant used

	Vars	Eq	Cancel Homogeneous Time	Cancel all τ and b Time
DAGS_1	52	104	1.3s	1.2s
DAGS_3	49	88	1.9	1.8s
DAGS_5	34	44	0.3s	0.3

However, in Table B.1, we can see that by canceling all the part of the system containing the variables τ and b, we get slightly faster time. Although it is interesting to always improve the time, this reduction is not really pertinent here.

Matricial Algebraic Systems and Application to the DAGS Cryptanalysis

The National Institute of Standards and Technology (NIST) launched a Call for Submissions in 2017 to find suitable algorithm candidates to be secure against quantum computers. Among the candidates was DAGS, an key encapsulation mechanism based on quasi-dyadic General Srivastava codes. In 2018 this scheme was attacked using an algebraic system, whose solving allows to find parts of the secret. We studied the modelisation that was done for this attack, and improved it in multiple ways. First, we can modify the input in order to shorten the time taken for the system solving, which also allowed us to be successful against a set of parameters that was unbroken before. Concurrently, the Gröbner basis computation was not behaving as it would for a generic system, showing more degree falls than what was expected. By comparing it to developments in the MinRank problem modelisation, we noticed that they were similar, giving us a model to follow for the methods to use and formulas linked to complexity. It appeared that the structure of DAGS was reducing the complexity of solving the modelisation system, and we went further to get even better results. We transformed our system by considering the minors as new variables, making the matrices involved smaller, like it is in the SupportMinors model. In DAGS case, this model behaves particularly well, and is able to compute a Gröbner basis efficiently.

Keywords : post-quantum cryptography - code-based cryptography - algebraic attack - MinRank problem - DAGS cryptosystem - Gröbner bases

---

Systèmes Algébriques Matriciels et Application à la Cryptanalyse de DAGS

Le National Institute of Standards and Technology (NIST) a lancé en 2017 un Appel à Soumissions pour trouver des algorithmes candidats pour sécuriser les données contre les ordinateurs quantiques. Le candidat DAGS est un mécanisme d'encapsulation de clé basée sur les codes de Srivastava quasi-dyadiques. En 2018, une attaque algébrique a été publiée, permet de retrouver une partie du secret du cryptosystème. Après étude de cette modélisation, nous avons essayé de l'améliorer de différentes façons. Nous avons d'abord modifié le système donné en entrée de l'agorithme de bases de Gröbner, ce qui nous a permis de raccourcir le temps de l'attaque ainsi que d'être efficace contre un ensemble de paramètres qui n'avait pas encore été cassé. Nous avons pu noter à ce moment que le calcul des bases de Gröbner n'avait pas le même comportement que pour un système générique : les chutes de degrés sont plus nombreuses. En comparant nos observations avec les récentes publications sur les systèmes modélisant le problème MinRank, nous avons remarqué qu'ils étaient similaires. Cela nous a permis d'avoir un modèle à suivre et des formules de base pour la complexité, tout en soulignant le fait que la structure du cryptosystème DAGS réduit cette même complexité. Nous avons continué les améliorations en considérant les mineurs comme de nouvelles variables, ce qui a pour avantage de réduire la taille des matrices utilisées, comme c'est le cas pour la modélisation SupportMinors. Dans le cas de DAGS, celle-ci se comporte particulièrement bien, et calcule une base de Gröbner de manière efficace.

Mots-clés : cryptographie post-quantique - cryptographie basée sur les codes - attaque algébrique - problème MinRank - cryptosystème DAGS - bases de Gröbner