HAL open science

Algebraic and Physical Security in Code-Based Cryptography

Frédéric Urvoy de Portzamparc

To cite this version:

Frédéric Urvoy de Portzamparc. Algebraic and Physical Security in Code-Based Cryptography. Other [cs.OH]. Université Pierre et Marie Curie - Paris VI, 2015. English. NNT : 2015PA066106 . tel-01187916

HAL Id: tel-01187916

https://theses.hal.science/tel-01187916v1

Submitted on 28 Aug 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ PIERRE ET MARIE CURIE - PARIS 6

ÉCOLE DOCTORALE EDITE

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université Pierre et Marie Curie - Paris 6
Mention : Informatique

Présentée par

Frédéric de Portzamparc

Sécurités algébrique et physique en Cryptographie fondée sur les codes correcteurs d'erreurs

Thèse dirigée par Jean-Charles Faugère
encadrée par Aline Gouget et Ludovic Perret
préparée au sein du Security Lab de Gemalto
et du Laboratoire d'Informatique de Paris 6 (UPMC)

Soutenue le 17 avril 2015 après avis des rapporteurs :

Daniel Augot	-	Directeur de Recherche Inria, CRI Saclay
Thierry Berger	-	Professeur, Université de Limoges

devant le jury composé de :

Daniel Augot	-	Directeur de Recherche Inria, CRI Saclay
Thierry Berger	-	Professeur, Université de Limoges
Alain Couvreur	-	Chargé de Recherche Inria, LIX
Jean-Charles Faugère	-	Directeur de Recherche Inria, CRI Paris-Rocquencourt
Aline Gouget	-	Docteur, Gemalto
Sylvain Guilley	-	Professeur, Institut Mines-Télécom
Ludovic Perret	-	Maître de Conférences, UPMC
Mohab Safey El Din	-	Professeur, Université Pierre et Marie Curie

Résumé

La cryptographie à base de codes correcteurs, introduite par Robert McEliece en 1978, est un candidat potentiel au remplacement des primitives asymétriques vulnérables à l'émergence d'un ordinateur quantique. Elle possède de plus une sécurité classique éprouvée depuis plus de trente ans, et permet des fonctions de chiffrement très rapides. Son défaut majeur réside dans la taille des clefs publiques. Pour cette raison, plusieurs variantes du schéma de McEliece pour lesquelles les clefs sont plus aisées à stocker ont été proposées ces dernières années.
Dans cette thèse, nous nous intéressons aux variantes utilisant soit des codes alternants avec symétrie, soit des codes de Goppa sauvages. Nous étudions leur résistance aux attaques algébriques et exhibons des faiblesses parfois fatales. Dans chaque cas, nous révélons l'existence de structures algébriques cachées qui nous permettent de décrire la clef secrète par un système non-linéaire d'équations en un nombre de variables très inférieur aux modélisations antérieures. Sa résolution par base de Gröbner nous permet de trouver la clef secrète pour de nombreuses instances hors de portée jusqu'à présent et proposés pour un usage à des fins cryptographiques. Dans le cas des codes alternants avec symétrie, nous montrons une vulnérabilité plus fondamentale du processus de réduction de taille de la clef.
Pour un déploiement à l'échelle industrielle de la cryptographie à base de codes correcteurs, il est nécessaire d'en évaluer la résistance aux attaques physiques, qui visent le matériel éxécutant les primitives. Nous décrivons dans cette optique un algorithme de déchiffrement McEliece plus résistant que l'état de l'art.

Abstract

Code-based cryptography, introduced by Robert McEliece in 1978, is a potential candidate to replace the asymetric primitives which are threatened by quantum computers. More generral, it has been considered secure for more than thirty years, and allow very vast encryption primitives. Its major drawback lies in the size of the public keys. For this reason, several variants of the original McEliece scheme with keys easier to store were proposed in the last years.
In this thesis, we are interested in variants using alternant codes with symmetries and wild Goppa codes. We study their resistance to algebraic attacks, and reveal sometimes fatal weaknesses. In each case, we show the existence of hidden algebraic structures allowing to describe the secret key with non-linear systems of multivariate equations containing fewer variables then in the previous modellings. Their resolutions with Gröbner bases allow to find the secret keys for numerous instances out of reach until now and proposed for cryptographic purposes. For the alternant codes with symmetries, we show a more fondamental vulnerability of the key size reduction process.
Prior to an industrial deployment, it is necessary to evaluate the resistance to physical attacks, which target device executing a primitive. To this purpose, we describe a decryption algorithm of McEliece more resistant than the state-of-the-art.

Contents

General Introduction . . . 3
Résumé . . . 17

I Preliminaries **31**

1 Coding Theory **33**
1.1 Code basics . . . 33
1.1.1 First definitions . . . 33
1.1.2 Operations on Linear Codes . . . 35
1.1.3 Code Permutation and Automorphism Group . . . 36
1.2 Generalized Reed-Solomon and Associated Codes . . . 37
1.3 Decoding Tools for Alternant codes . . . 39
1.3.1 Alternant Decoder . . . 40
1.3.2 Patterson Method for Decoding Binary Goppa Codes . . . 43

2 Code-Based Cryptography **47**
2.1 McEliece Encryption Scheme . . . 48
2.2 Syndrom Decoding Problem and Information Set Decoding . . . 49
2.3 The Decoder Recovery Problem . . . 50
2.3.1 The Sidelnikov-Shestakov Attack . . . 50
2.3.2 The Goppa Decoder Recovery Problem . . . 52
2.4 The Goppa Distinguishing Problem . . . 54

3 Gröbner Bases **57**
3.1 Basic Definitions . . . 58
3.2 General Strategy for solving polynomial system . . . 61
3.2.1 Complexity Asymptotics . . . 65
3.2.2 Degree of regularity in the algebraic modelling of alternant codes . . . 66
3.3 Gröbner Bases and Modelling Strategy . . . 67

II Contributions **69**

4 Algebraic attacks against McEliece-like schemes **71**
4.1 Algebraic Descriptions . . . 72
4.1.1 Algebraic Description of Goppa Codes . . . 73
4.1.2 Algebraic Description of Binary Goppa Codes . . . 73
4.2 Structural Elimination . . . 74
4.3 Summary of the proposed Algebraic modellings . . . 75

4.4 Complexity Bounds & Asymptotics . . . 76
4.4.1 Full-Support Codes . . . 77
4.4.2 Non Full-Support codes . . . 77

5 Alternant Codes with Non-trivial Permutation Groups 81
5.1 Alternant Codes with Symmetries in Cryptography. . . . 83
5.2 Classification . . . 85
5.2.1 GRS Codes with Non-Trivial Permutation Groups . . . 85
5.2.2 Alternant Codes with Non-Trivial Permutation Groups . . . 87
5.3 Link with the known constructions . . . 88
5.3.1 Quasi-Cyclic Alternant Codes ([BCGO09]) . . . 89
5.3.2 Quasi-Dyadic/Monoidic Alternant Codes ([MB09, BCMN10, BLM11]) . 89
5.3.3 Generalization of Quasi-Monoidic Alternant Codes. . . . 89
5.4 The Folded Code . . . 92
5.5 Folding QC and QM GRS Codes . . . 93
5.5.1 Affine-Invariant Polynomials . . . 93
5.5.2 Folding QC and QM GRS Codes . . . 95
5.6 Folding QC and QM Alternant and Goppa codes . . . 97
5.7 Technical Results . . . 101
5.7.1 Proof of Proposition 5.17 . . . 101
5.7.2 Proof of Proposition 5.18 . . . 103

6 Algebraic cryptanalysis of QM **McEliece 105**
6.1 Iterated Folded Code . . . 107
6.2 Application to the Cryptanalysis of QM alternant codes . . . 110
6.2.1 Full Cryptanalysis Strategy . . . 110
6.2.2 Extension to all signature parameters . . . 110
6.2.3 Practical experiments . . . 113
6.3 Conclusion about the symmetric Alternant Codes . . . 115

7 Algebraic cryptanalysis of Wild McEliece 117
7.1 New Algebraic Modelling . . . 120
7.1.1 First Simplification of the System (FOPT) . . . 120
7.1.2 The Algebraic Modelling $\mathcal{W}_{q,a}(\mathbf{Z})$ . . . 122
7.2 Solving the Goppa Decoder Recovery Problem . . . 124
7.2.1 Boosting the Resolution of $\mathcal{W}_{q,a}(\mathbf{Z})$ . . . 124
7.2.2 Disentanglement of the System Solutions . . . 125
7.2.3 Sidelnikov-Shestakov Adapted To Recover the Goppa Polynomial . . . 127
7.2.4 Recovery of the Incognito Polynomial by Solving a Linear System . . . 129
7.3 Application to Wild McEliece & Incognito . . . 130
7.3.1 A property of Wild Goppa codes over non-prime base fields . . . 130
7.3.2 Practical experiments & Broken Challenges . . . 133
7.4 Conclusion . . . 136

8 Toward a Secure Implementation of McEliece 137
8.1 Timing Attacks against McEliece Decryption . . . 139
8.1.1 Plaintext-Recovery Attacks . . . 139
8.1.2 Secret decryption key recovery attacks . . . 143
8.2 Extended euclidean Algorithm with constant flow . . . 148
8.2.1 Unrolling Euclidean divisions. . . . 149

8.2.2 Regular polynomial shift pattern. 150
8.2.3 Complete Regular Flow EEA. 150
8.3 Proofs of completeness 154
8.3.1 Proof of Proposition 8.5. 154
8.3.2 Proof of Proposition 8.6. 155
8.3.3 Proof of Proposition 8.7. 155

General Introduction

Context

Cryptology studies the methods to protect secret information. Historically, cryptology was a major concern for governments during wars, as the information exchanged between the command center and the mission theater is critical for the success of the operation. Throughout history, militarians have used cryptology to protect their communication. A famous and old example is the method used by Julius Caesar to protect his private correspondence. A more recent example is provided by the Enigma machine used by the German militaries from 1920 until the end of World War II. It is a common belief that the discovery by Polish and British scientists of a method to circumvent the protection applied to messages by Enigma shortened the war by two years.

Nowadays, the use of cryptology is not confined to military communication anymore. The need for secure communication concerns every one, as the electronic mail and transactions are sent through public networks. We also need to protect our identities when we identify on remote services. Willingly or not, smartphone owners store a considerable amount of data on their private lives on a device which is constantly connected to networks whose security is unclear. Dealing with these problems, amongst others, is in the scope of modern cryptology.

A milestone was reached in cryptology in 1976 with the advent of *public-key cryptography*. Before this date, for Alice and Bob to communicate secretly, they had to share a common secret prior to their message exchange. For instance, Caesar had to transmit *securely* a secret shift to the chosen recipient. This secret exchange becomes highly unconvenient when a growing number of entities wish to communicate. The breakthrough described in 1976 by Diffie and Hellman consisted in describing how Alice and Bob can publicly agree on a value which will be known by only the two of them (a *secret*), so that even someone able to intercept all their exchanges cannot deduce information.

Cryptanalysis is the field of cryptology dedicated to bypassing the methods used to protect information and recover data that were intended to remain secret for unauthorized users. The process of recovering a specific hidden information is often referred to as an *attack*. Evaluating the feasability of an attack is of crucial importance to guarantee the security of protected data. Indeed, when selling or using a device implementing cryptographic functions, it is mandatory to estimate the resistance of the device to all the known attacks. This is what we strive to do in this thesis, in a specific domain of cryptography that we will describe.

Error-Correcting Codes

Error-correcting codes are mathematical tools originally aiming at protecting data from transmission errors. The general idea is, when sending a message, to add some redundancy to this message so that, if some bits are alterated during the transmission the receiver can correct the errors and recover the original message. The function adding redundancy is called

the *encoding*, the one correcting the errors in the received message is the *decoding*. In practice, the decoding is successful if the number of bits alterated is lower than a threshold called the *correction capacity* of the encoding.

Error-correcting codes are present everywhere in digital transmissions: payloads on internet and messages sent by satellites through space are protected against transmission errors, music stored on compact discs is protected from scratches... A large variety of encoding functions was developped according to the industrial needs: for instance, encoding with high correction capacity for satellites, very fast decoding functions for real-time use (*e.g.* when reading a CD). Many eamples can be found in [MS86, PBH98].

A wildspread family of codes is the family of *linear codes*, thus called because the set of all the possible encoded messages form a linear vector space over a finite field (*i.e.*, they form a subspace of $\mathbb{F}_q^n$ for some prime power q and integer n, which is called the *length* of the code). They can be conveniently represented by giving a *basis* of the vector space (for instance under the form of a matrix). Let us assume that the messages are strings of k bits, we write them as row vectors $\mathbf{m} = (m_0, \ldots, m_{k-1}) \in \mathbb{F}_2^k$. A linear error-correcting code admits a simple encoding function represented by a $k \times n$ matrix $\mathbf{G}$, that can be picked of the form $\mathbf{G} = (\mathbf{I}_k|\mathbf{A})$, with $\mathbf{I}_k$ the identity matrix of size $k \times k$, and $\mathbf{A}$ a $k \times (n-k)$ matrix. The encoding function is :

$$\mathbf{m} \in \mathbb{F}_2^k \mapsto \mathbf{mG} = \mathbf{m}(\mathbf{I}_k|\mathbf{A}) = (\mathbf{m}, \mathbf{b}) \in \mathbb{F}_2^n$$

where $\mathbf{b}$ is the redundancy. Encoding simply consists in making a linear combination of elements of the basis. On the contrary, decoding is more complex in general. Suppose that an encoded message is received with some errors, we write the received message as $\mathbf{m}' = \mathbf{mG} + \mathbf{e}$, where $\mathbf{e}$ is a vector of $\mathbb{F}_2^n$ whose entries are all zero except in the positions where tranmission errors occured. Decoding reduces to solving the system

$$\mathbf{m}' = \mathbf{mG} + \mathbf{e}$$

where $\mathbf{m}', \mathbf{G}$ are known and $\mathbf{m}, \mathbf{e}$ are unkown. This is referred to as *solving a linear system with errors*, which was proved to be NP-hard (in [BMvT78, LDW94]). One of the goals of the theory of linear error-correcting codes is to find special linear spaces for which the decoding can be solved efficiently in some cases (*e.g.* when the number of alterated bits is bounded). Chapter 1 of this thesis gives the example of *alternant codes* and (binary) Goppa codes, which are the linear codes that we focus on in our work.

Code-based Cryptography

A first link between error-correcting codes and cryptography was made in 1978 by McEliece in [McE78]. The idea is that, for certain families of error-correcting codes endowed with an efficient decoding, applying the decoding procedure requires to know a specific mathematical element used to build the linear space, but which is hard to recover when only a basis is known. Therefore, an encoded message in which errors are introduced becomes unreadable to anyone ignoring the hidden element. In cryptographic terms, the encoding function is a *trapdoor one-way function*, that is, a function difficult to invert without knowing the private trapdoor. We describe a public-key encryption scheme as follows: the encoding function (described by a basis of the linear space, *e.g.* a matrix $\mathbf{G}_{pub}$) is the public-key, the encryption function consists in encoding a message $\mathbf{m}$ and deliberately introducing errors. The secret key is the hidden element used to decode efficiently, and decryption consists precisely in applying the decoding procedure.

$$\overset{\text{plaintext } \mathbf{m}\in\mathbb{F}_q^k}{\begin{pmatrix} 0,\ldots,1 \end{pmatrix}} \overset{\text{public key } \mathbf{G}_{pub}\in\mathbb{F}_q^{k\times n}}{\begin{pmatrix} \mathbf{G}_{pub} \end{pmatrix}} + \overset{\text{error } \mathbf{e}\in\mathbb{F}_q^n}{\begin{pmatrix} 1,0,\ldots,0,1 \end{pmatrix}} = \overset{\text{ciphertext } \mathbf{c}\in\mathbb{F}_q^n}{\begin{pmatrix} 1,0,\ldots,1,1 \end{pmatrix}}$$

Principle of code-based encryption

A central question in code-based cryptography consists in finding error-correcting codes with such a hidden element required for decoding. The security of the encryption relies both on the hardness of decoding without the required element and the difficulty of recovering the trapdoor knowing only the encoding function. The family of codes proposed by McEliece, namely the *binary Goppa codes*, are still considered to satisfy those criteria today. They belong to the family of alternant codes, which are also considered as secure. In Chapter 2, we present known methods to recover the trapdoor for those codes, and detail their computational cost.

After this proposition of trapdoor one-way function, which is known as McEliece encryption scheme, other cryptographic primitives using linear codes were introduced. They share the property that their security relies on the hardness of decoding a linear code, and we gather in code-based cryptography all the primitives with this feature. Some of the most famous ones are Niederreiter encryption scheme [Nie86], Stern's authentication scheme [Ste93] and the CFS signature scheme (for Courtois, Finiasz and Sendrier) [CFS01]. Relying on other assumptions than number-theory problems such as the Discrete Logarithm Problem and integer factorization is a very positive characteristic of code-based primitives. Indeed, thoses are only assumptions, and algorithmic progresses are always possible, so that it is preferable to have cryptographic primitives whose security is backed up by as many different assumptions as possible.

Moreover, no quantum algorithm is known to decode a random linear code in polynomial time, contrary to the Discrete Logarithm Problem and the integer factorization problem which Shor's algorithm [Sho94a, Sho94b] solves in (probabilistic) time $O(\log(n)^3)$ with a quantum computer (where $\log(n)$ is the size of the input). Actually, should a quantum computer arise, there exists a long list of public-key cryptosystems that would not be usable anymore in a post-quantum era (Diffie-Hellman key exchange, RSA encryption, ECDSA signatures, elliptic and hyper-elliptic curves cryptography...). The possibility of the emergence of a quantum computer has grown sufficiently for standard organizations to start preparing the standardization of some post-quantum primitives including code-based cryptosystems [1].

Recent Trends in Code-based Cryptography

As the reflect of a growing interest for code-based cryptography, several axes of research aiming at designing code-based cryptographic primitives suitable and practical for applications were explored in the last years.

Reduction of the public-key storage cost

Since 1978, code-based cryptography has not been widely deployed in practice. The main reason is that it does not compare favorably to the other public-key primitives (*e.g.* RSA) in terms of key size. For instance, for a security level of 2^{128}, the public key used by the McEliece

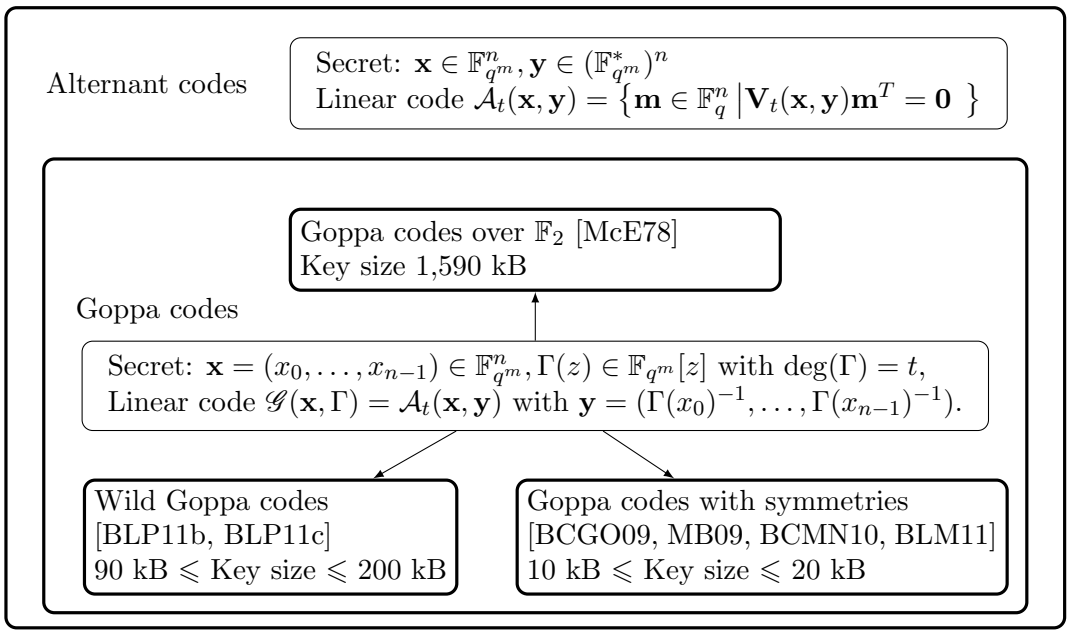
1. `http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/Quantum_Safe_Whitepaper_1_0_0.pdf`

cryptosystem is 60 times bigger than RSA key. Many proposals have been made in order to reduce the key size since 1985, *e.g.* [Gab85, Nie86, Sid94, JM96, MRS00, Gab05, BL05, BCGO09, MB09, BLM11, BLP11b, BLP11c, Per12a, MTSB13]. Determining the security of new propositions is crucial, and many (but not all) of the previously mentionned ones were broken [SS92, Wie06b, Ove08, FOPT10a, CMCP14].

In this thesis, we focus on two different families of codes with reduced key sizes. They are both special cases of a large family of codes called the *alternant* codes, which also encompass the binary Goppa codes proposed by McEliece. Namely, we will adress the security of alternant (and Goppa) codes *with symmetries* proposed in [BCGO09, MB09, BCMN10, BLM11], and on the *Wild Goppa codes*, introduced in [BLP11b, BLP11c]. To define all these codes, we introduce the following Vandermonde-like matrices:

$$\mathbf{V}_t(\mathbf{x},\mathbf{y}) = \begin{pmatrix} y_0 & \cdots & y_{n-1} \\ y_0x_0 & \cdots & y_{n-1}x_{n-1} \\ \vdots & & \vdots \\ y_0x_0^{t-1} & \cdots & y_{n-1}x_{n-1}^{t-1} \end{pmatrix}, \tag{1}$$

for $\big(\mathbf{x} = (x_0,\ldots,x_{n-1}), \mathbf{y} = (y_0,\ldots,y_{n-1})\big) \in \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n$. We sum up in the following figure the sub-families of the alternant codes of particular interest for this thesis.



Sub-families of alternant codes considered in this thesis. Key sizes are indicated for encryption schemes with security 2^{128}.

Algebraic Cryptanalysis in Alternant Code-Based Cryptography

For long, the cryptanalysis of code-based scheme has reduced to generic message-recovery attacks, which consist in decoding a random linear code. These methods are referred to as ISD (standing for *Information Set Decoding*). Very few methods were known for key-recovery, and, to our knowledge, none execpt [Sen00, LS01]. New methods of emerged recently. They belong

to the category of *algebraic cryptanalyses*. Algebraic cryptanalysis is a method of attack that links the secret targeted by the attacker with the resolution of a system of multivariate algebraic equations. To do so, the first step of an algebraic cryptanalysis consists in finding an algebraic modelling, that is, a way of representing the secret elements as unknowns of a polynomial system. The second step consists, if possible to solve the system, or at least to estimate its complexity of resolution. Solving efficiently a non-linear system of multivariate equations is a fundamental problem, referred to as PoSSo, that appears in a large variety of scientific fields (cryptology, mechanics, biology...). It was proved to be NP-hard over a finite field ([GJ90]). However, in cryptology, the systems considered are rarely random, as they are derived from applications, so that they have a (sometimes hidden) structure. In many examples, the algebraic systems turned out to be solvable in practice (*e.g.* [FJ03, CM03, FLdVP08, FS10, FOPT10a, BFP11]).

To solve polynomial systems over finite fields, several methods exist (exhaustive search, SAT solvers ...). In this thesis, we use Gröbner bases ([Buc65]), which are introduced in Chapter 3. It has the advantage of providing tools to analyze the complexity of the associated resolution algorithms, which is useful to deduce security levels in cryptography. Also, the analysis of Gröbner bases computation algorithms can be a fruitful source of information on a potential hidden structure of the modelled object.

Regarding alternant codes, an algebraic modelling was introduced in [FOPT10a]. Let $\mathbf{X} = (X_0, \ldots, X_{n-1})$ and $\mathbf{Y} = (Y_0, \ldots, Y_{n-1})$ be variables in $\mathbb{F}_{q^m}$ corresponding to the unkown secret key, and the $g_{i,j}$'s be the entries of a $k \times n$ generator public matrix. Then, $\mathbf{X}$ and $\mathbf{Y}$ satisfy:

$$\Big\{ g_{i,0} Y_0 X_0^\ell + \cdots + g_{i,n-1} Y_{n-1} X_{n-1}^\ell = 0 \ \mid \ i \in \{0, \ldots, k-1\}, \ell \in \{0, \ldots, t-1\} \Big\}. \tag{2}$$

In practice, for cryptographic parameters, this modelling does not allow the recover the secret elements of a plain McEliece cryptosystem because the cost of the resolution of the system (2) is too high with the current computation capacity. However, it yielded in [FGO$^+$13] an algorithm to determine in polynomial time if a given matrix is the generator matrix of an alternant code (of high dimension). It was conjectured that such efficient tool, called a *code distinguisher*, did not exist [Sen02]. Also, thanks to this modelling, a key-recovery attack was designed against a sub-family of alternant codes with symmetries, but the codes defined over a prime field were not threatened at all. This was therefore used as a protection to propose codes with symmetries immune to this algebraic attack [BCMN10, BLM11].

Finding Secure and Efficient Practical Implementations

An important step towards the use of code-based cryptography in the real world consists in designing secure implementations. Indeed, when implementing cryptographic functions, it is necessary to avoid introducing vulnerabilities against so-called *side-channel attacks*.

Side-channel attacks are methods of recovering secret information introduced by Kocher in [Koc96]. In this article, it was showed that an attacker measuring the execution time of an RSA decryption could recover the secret key. The idea is that, with a standard implementation, the number of operations to perform, and thus the execution time, depend on the bits of the secret key. The attack consists in guessing possible secret keys and check if the targeted device has the expected behaviour. Since, other physical emanations have been used to practically attack devices (*e.g.* power consumption [KJJ99, ÖOP03], electro-magnetic field [GMO01] and acoustic waves [GST14]). Side-channel analysis has become a field of research in itself, and it has become mandatory to implement the algorithms using a secret element in

a way to avoid those attacks. The purpose is that the physical emanations of the device be completely independant of the secret information. If it is not the case, we say that there is a *leakage*, and the purpose of a physical attack is to measure and exploit it. The modifications brought to the implementation in order to eliminate this leakage are called *counter-measures*.

This work is specific to each mathematical operation (*e.g.* exponentiation of a cyclic group element or an elliptic curve point, vector addition, matrix-vector product...). There exists a rich literature on this topic for the widely used primitives such as RSA, but for code-based cryptography it is only recently that researchers began to investigate the weaknesses of too naive implementations (for instance in [SSMS09, AHPT11, Str10b, Str13]), and make propositions of hopefully more resistant ones [Hey11, Str10a, SWM$^+$10, BCS13].

Some articles made propositions towards secure and efficient implementations of code-based primitives, for instance [CGP08, Str10a, Bis10, SWM$^+$10, Hey11, LS12, BCS13]. Some of them take into account various potentially exploitable physical leakages, some of them do not and focus on the efficiency. In parallel, several works on attacking implementations of McEliece decryption emerged [SSMS09, AHPT11, Str10b, Str13]. Those articles identify leakages and propose most of the time counter-measures against them. However, no clear counter-measure is proposed to protect from the timing attack of [Str13].

Guarantee the Conformity with Security Models

Security models are abstractions of cryptographic primitives which allow to measure and bound the quantity of information that an attacker can hope to retrieve. They are abstractions in the sense that they do not consider the mathematical properties of the function but only its cryptographic features: for instance, for an encryption primitive, security models try evaluate the capacity of an attacker to link a ciphertext and a corresponding plaintext. On the contrary, to prove that a cryptographic primitive satisfies a security model, it is necessary to consider its mathematical properties. Regarding code-based cryptography, a series of work aimed at evaluating which security requirements are satisfied by code-based primitives and how to modify them so that they fit into security models, *e.g.* in [Ber97, KI01, NIKM08, NC11, Per12b, DDMQN12].

Contributions

For those recent proposals (alternant codes with symmetries and Wild McEliece), we revealed unkown properties of the schemes which turned to be weaknesses. Indeed, thanks to these properties, we introduced new key-recovery attacks allowing practical breaks of many parameters that had been proposed for cryptographic use. Those new attacks will have to be taken into account in the future to propose secure parameters for related schemes.

Algebraic Cryptanalysis in Alternant Code-Based Cryptography.

In Chapter 4 of this thesis, we study an algebraic attack against generic alternant codes provided by the modelling (2). We improve this algebraic modelling to describe more precisely famous subfamilies of alternant codes. We also introduce new modellings, where the blocks $\mathbf{X}$ and $\mathbf{Y}$ of (2), containing n variables each, are replaced by blocks with $n-k$ variables. We provide complexity bounds for the resolution of the associated algebraic systems. To do so, we assume that we found by exhaustive search the correct values of enough variables to be able to recover the other ones by solving a linear system. By this method, we show that the number of operations necessary to recover the secret elements of an alternant code is bounded

above by $q^{m^2 t} \cdot \mathrm{poly}(n)$. We provide a comparison of those complexity bounds with the other key-recovery attack on Goppa codes (namely the Support Splitting Attack). It shows that, asymptotically, the algebraic attack that we describe is the best one for codes whose support contains significantly fewer elements than the finite field from which they are picked (that is, $n \ll q^m$).

McEliece Schemes Using Alternant Codes with Symmetries.

Alternant codes with symmetries refer to alternant codes admitting a generator matrix made of symmetric blocks. Such matrices can be fully described by storing only a reduced number of entries. On the Figure on page 9, storing only 16 elements a_i is sufficient to fully know a matrix with 64 entries. Those codes were introduced by different authors [BCGO09, MB09, BCMN10, BLM11] in order to design encryption schemes whose public key has a reduced storage cost. Indeed, the cryptosystems proposed in [MB09] allowed a reduction factor between 22 and 117 compared to a binary Goppa code with same security level. Faugère, Otmani, Perret and Tillich (FOPT) showed in [FOPT10a] that the generic alternant modelling (2) could be adapted to those codes with symmetries. However, solving the resulting algebraic modelling was not feasible for all the codes. Consequently, practical key-recoveries were possible for all the parameters from [BCGO09] and some from [MB09], but not for the parameters from [BCMN10, BLM11].

In Chapter 5, we gather all the alternant codes with symmetries proposed so far under a common feature (namely, they have a non-trivial automorphism group). Thanks to this property, we give a framework to build, from an alternant code without symmetry, an alternant code with symmetry. For istance, we show that all the codes proposed in [BCGO09, MB09, BCMN10, BLM11] can be obtained by sampling supports $\mathbf{x} \in \mathbb{F}_{q^m}^n$ globally stable by an affine-map $\phi : z \mapsto az + b$ (with $(a, b) \in \mathbb{F}_{q^m}^* \times \mathbb{F}_{q^m}$). Then, we show that, for those codes, it is possible to somehow *reverse* it. To do so, we build, from the public matrix with symmetric blocks, a code where each block of entries is replaced by the sum of the entries of the block. We call this construction the *folding*, and the resulting code the *folded* code.

$$\left(\begin{array}{cccc|cccc}
a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\
a_1 & a_0 & a_3 & a_2 & a_5 & a_4 & a_7 & a_6 \\
a_2 & a_3 & a_0 & a_1 & a_6 & a_7 & a_4 & a_5 \\
a_3 & a_2 & a_1 & a_0 & a_7 & a_6 & a_5 & a_4 \\
\hline
a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\
a_9 & a_8 & a_{11} & a_{10} & a_{13} & a_{12} & a_{15} & a_{14} \\
a_{10} & a_{11} & a_8 & a_9 & a_{14} & a_{15} & a_{12} & a_{13} \\
a_{11} & a_{10} & a_9 & a_8 & a_{15} & a_{14} & a_{13} & a_{12}
\end{array}\right) \xrightarrow{\text{Folding}} \left(\begin{array}{c|c}
\begin{array}{c} a_0 + a_1 \\ +a_2 + a_3 \end{array} & \begin{array}{c} a_4 + a_5 \\ +a_6 + a_7 \end{array} \\
\hline
\begin{array}{c} a_8 + a_9 \\ +a_{10} + a_{11} \end{array} & \begin{array}{c} a_{12} + a_{13} \\ +a_{14} + a_{15} \end{array}
\end{array}\right)$$

Folding of an 8×8 matrix with symmetries, made of four 4×4 symmetric blocks, into a 2×2 matrix.

The weakness lies in the fact that this folded code is also an alternant code, and has almost same private elements as the original code. We proved this by showing that, for ϕ the affine map used to build the secret support and $\alpha \in \mathbb{F}_{q^m}^*$, there exist polynomials $A_{\phi,\alpha}, B_{\phi,\alpha} \in \mathbb{F}_{q^m}[z]$ such that any polynomial $P \in \mathbb{F}_{q^m}[z]$ satisfying $P(\phi(z)) = \alpha P(z)$ can be written under the form $P(z) = A_{\phi,\alpha}(z) Q(B_{\phi,\alpha}(z))$, where $Q \in \mathbb{F}_{q^m}[z]$ is of degree lower than P. This leads us to our central contribution on alternant codes with symmetries, summed up below.

Theorem (informal, see Theorem 5.21 on page 98)**.** *Let $\mathscr{C}$ be an alternant code of length n,*

with secret $(\mathbf{x}, \mathbf{y}) \in \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n$*, whose generator matrix is made of symmetric blocks of size* $t \times t$ *as in [BCGO09, MB09, BCMN10, BLM11]. Then, the* folded *code of* $\mathscr{C}$ *is an alternant code* $\mathscr{C}'$ *of length* n/t*, with secret* $(\mathbf{x}', \mathbf{y}') \in \mathbb{F}_{q^m}^{n/t} \times \mathbb{F}_{q^m}^{n/t}$*. There is an explicit relation between* $(\mathbf{x}', \mathbf{y}')$ *and* $(\mathbf{x}, \mathbf{y})$*, that is the secret* $(\mathbf{x}, \mathbf{y})$ *defining* $\mathscr{C}$ *can be recovered from the knowledge of* $(\mathbf{x}', \mathbf{y}')$*.*

Therefore, the secret key can be recovered from the folded code, which has no more symmetries. We prove that those McEliece schemes with compact representations have intrinsically same key-security as a McEliece scheme *without symmetry* but with same small key size.

In Chapter 6, we combine the weakness of alternant codes with symmetries that we exhibited with our refined algebraic modellings. We obtain an algebraic attack against McEliece using alternant and Goppa codes whose support $\mathbf{x}$ is stable by maps of the form $\phi : z \mapsto z + b$ with $b \in \mathbb{F}_{q^m}^*$, which are those used from all [MB09, BCMN10, BLM11]. We restricted to this case because the other possible affine maps (namely $\phi : z \mapsto az + b$ with $a \neq 1$), considered in [BCGO09], were fully broken in [FOPT10a]. On the contrary, this attack was structurally inefficient for the parameters of [BCMN10, BLM11]. Practical results show that the attack is a fatal threat for codes with high dimensions k (that is, close to n), which is always the case for the parameters in signature schemes. For encryption schemes, the attack is less efficient for codes with small dimensions, but those codes are more vulnerable to generic message-recovery attacks. Therefore, it is not sure that secure encryption parameters can be found.

Code Parameters (q, m, t, n)	This thesis	Previous attack	Security level	Ref
$(2^4, 4, 128, 4096)$	0.010 s.	7.1 s	128	[MB09]
$(2^2, 8, 64, 3584)$	0.040 s.	1,776.3 s	128	[MB09]
$(2, 16, 32, 4864)$	18 s.	N.A.	128	[MB09]
$(2, 14, 13, 16368)$	3.0 s	N.A	81	[BCMN10]
$(2, 14, 14, 16368)$	3.0s	N.A.	84	[BCMN10]
$(2, 15, 12, 32752)$	5.4s	N.A	82	[BCMN10]
$(3, 11, 9, 177048)$	3.4 s	N.A.	80	[BLM11]
$(5, 8, 15, 390495)$	82 s	N.A.	128	[BLM11]
$(241, 3, 241, 964)$	0.020 s	N.A.	112	[BLM11]

Key-recovery timings for alternant codes with symmetries (N. A. means Not Applicable).

Cryptanalysis of Wild McEliece

Chapter 7 of this thesis is dedicated to Wild McEliece and Wild McEliece Incognito, which are code-based encryption schemes described by Bernstein, Lange and Peters in [BLP11b] and [BLP11c]. The encryption function is the same as for McEliece, but the linear codes which are used have smaller bases than those used for McEliece encryption, so that the public key can be up to 16 times smaller. Those codes, which belong to the category of alternant codes, can be seen as a generalization of the Goppa codes over $\mathbb{F}_2$ proposed by McEliece to other fields $\mathbb{F}_q$ with $q > 2$. They were called *wild Goppa codes* by the designers of the schemes. Our results show that, thanks to the specificities of those Wild Goppa codes, the secret key can be modelled by an algebraic system containing fewer variables compared to the modelling of generic alternant codes (2). If the $g_{i,j}$'s are the entries of a $k \times n$ generator matrix of the public Wild Goppa code, and $\mathbf{Z} = (Z_0, \ldots, Z_{n-1})$ are unkowns, we prove that an attacker able to solve the following system:

$$\left\{ g_{i,0} Z_0^\ell + \cdots + g_{i,n-1} Z_{n-1}^\ell = 0 \ \mid \ i \in \{0, \ldots, k-1\}, \ell \in \{1, \ldots, q-1\} \right\}, \tag{3}$$

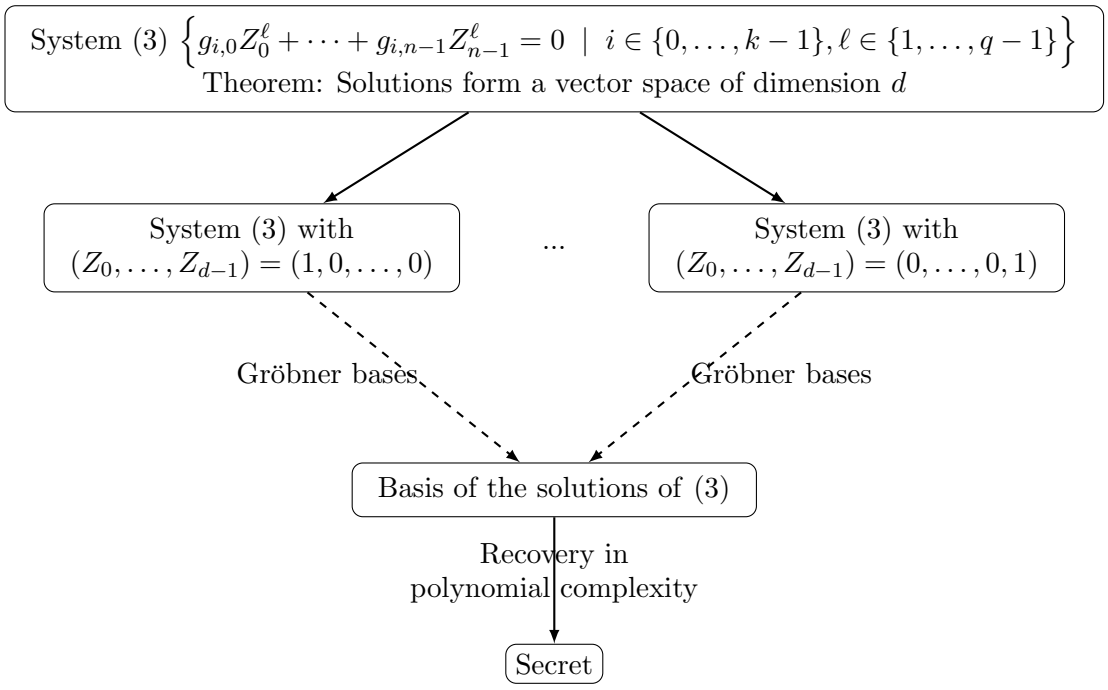
can recover the private key.

This was done by proving that the solutions of system (3) form a union of vector spaces, which is a unusual situation for a non-linear algebraic system. We exploit this in the resolution phase: since we are not looking for a specific linear combination but just a basis of the solutions, we can fix arbitrarily coordinates of the desired vector. Once the system solving phase is over, we designed a method to recover the secret from the solutions of the algebraic system in (heuristic) polynomial time. In essence, we proved the following result.

Theorem (informal, see Theorem 7.6 on page 122 and Theorem 7.15 on page 129)**.** *Let $\mathscr{C}$ be a Wild Goppa code over $\mathbb{F}_q$ and $g_{i,j}$ be the entries of a $k \times n$ generator matrix. Then,*

1. *the solutions of* (3) *form a union of vector spaces.*
2. *Given a basis of one of those vector spaces, the secret elements of $\mathscr{C}$ can be recovered in heuristic polynomial time.*

We deduce the following strategy of cryptanalysis.



Key-recovery strategy for Wild McEliece

More broadly, our compact algebraic modelling can be applied in a more general context than Wild McEliece, namely as soon as the secret polynomial $\Gamma(z)$ has a multiple factor. This should encourage designers to pick irreducible polynomial for Goppa codes.

We mounted a practical attack using this framework against wild Goppa codes over non-prime fields by revealing and exploiting extra-weaknesses of those codes. We show that, for a wild Goppa code defined over $\mathbb{F}_{p^s}$ with p prime and $s > 1$, we can replace the system

(3) (containing $(p^s-1)mt-t$ variables) by a variant which contains fewer than $(p-1)mst$ variables, and apply the same-key recovery strategy. This impacts the security of many parameters proposed for cryptographic use with a security level of 2^{128} by the designers in the articles [BLP11b, BLP11c]. They also proposed on a website a series of challenges with growing difficulty. For several field sizes, we can solve the hardest challenges proposed online.

Code parameters (q,m,t,u,n)	Key Size (kB)	Claimed Security	Generic modelling		**Our modelling**	
			Unk.	Res. Time	Unk.	Res. Time
$(32,2,3,24,852)$	90	2^{130}	462	∞	**18**	0.6s
$(27,3,2,42,1500)$	204	2^{128}	564	∞	**26**	0.9s
$(27,3,5,0,1700)$	304	2^{158}	780	∞	**65**	1h 59min
$(25,3,3,25,1206)$	55	2^{117}	582	∞	**57**	1h 2min
$(16,3,6,16,1328)$	160	2^{125}	636	∞	**54**	36h 35min
$(9,3,6,14,728)$	40	2^{81}	372	∞	**54**	25h 13min

Number of variables and resolution time (with Magma's F_4 implementation [BCP97a] of our algebraic modelling of some wild Goppa codes defined over $\mathbb{F}_q$ with secret polynomial $\Gamma(z)=f(z)g(z)^{q-1}\in\mathbb{F}_{q^m}[z]$ with $t=\deg(g)$ and $u=\deg(f)$.

Conclusion on the Algebraic Cryptanalyses against Goppa Codes

Throughout this thesis, we design our cryptanalyses by exploiting features of alternant and Goppa codes which are not generic. It is important to emphasize that none of those codes were random alternant or random Goppa codes. To sum up, we took part of the following additional properties:

— the existence of algebraic relations on the secret support and multipliers of alternant codes,
— the non-irreducibility of the Goppa polynomial of a Goppa code, because of multiple or different irreducible factors (except in the binary case),
— the fact that the public code is defined over a non-prime field.

Binary Goppa codes with irreducible Goppa polynomial meet none of those properties. The fact that each modification could be turned into a weakness should encourage designers to stick to them.

Our results also illustrates how studying the behaviour of algebraic modellings can be enlightening on the modelled mathematical object.

Let us take the example of alternant codes with symmetries. [FOPT10a] gave an algebraic modelling for the symmetric alternant codes obtained thanks to maps of the form $\phi: z\mapsto z+b$ (with $b\in\mathbb{F}_{q^m}^*$). It consists in a non-linear algebraic system which is very *overdetermined*, *i.e.* it contains more equations than unknowns. For a random overdetermined algebraic system, the probability that it admits solutions is very low. As the system here describes a secret key corresponding to the public Goppa code, it necessarily has solutions (but few). However, we observed the following property: after fixing a specific small subset of variables to the value 0, the obtained system still has solutions. Yet stranger, for another subset of variables, the solutions are the same whether the specific set of variables is fixed to zero or not. This property, unexpected and puzzling, is explained by understanding that fixing those values to 0 transforms the algebraic modelling of the public code into the algebraic modelling of its folded code.

Our work on Wild McEliece provides another example of code structure revealed by the study of the algebraic modellings. One can show that, for the $g_{i,j}$'s,and $\mathbf{Y}$ defined as in System (2), we can write an overdetermined system of *linear* equations because of the specificities of Wild Goppa codes. Indeed, when the entries $g_{i,j}$ of the public matrix $\mathbf{G}$ belong to $\mathbb{F}_q$ with $q = p^s$, it holds that:

$$\left\{ g_{i,0}^{p^u} Y_0 + \cdots + g_{i,n-1}^{p^u} Y_{n-1} = 0 \ \mid \ i \in \{0, \ldots, k-1\}, 0 \leqslant u \leqslant s-1 \right\}. \tag{4}$$

If the system (4) is full-rank, this is a very favorable situation for an attacker. The secret can then be recovered in polynomial time. However, we observed that these linear equations always has a rank defect with very specific form (namely $n - (p-1)mst$ for $q = p^s$). This rank defect could actually be linked with the dimension of a Wild Goppa code related to the public code, and we ended up with an unexpected result on the structure of the public code (Theorem 7.16).

Contribution Towards a Secure Implementation of McEliece Cryptosystem

Our contribution in the securization of the implementation of code-based primitives consisted in proposing a way of implementing the decryption in McEliece in a way which resists the so-called *side-channel attacks* better than the previous propositions. In Chapter 8 of this thesis, we show that the attack of [Str13] can be improved, so that a reliable counter-measure is required. We describe such counter-measure. It consists in an implementation of the extended Euclidean Algorithm that, once the public parameters are fixed, uses a fixed number of loops, each one containing a fixed number of operations. Doing so, all timing leakages are avoided. Therefore, we bring an important element towards a secure implementation of McEliece encryption scheme, and more generally in any primitive requiring the decoding of an alternant code.

Perspectives

Analysis of the Algebraic Modellings

In this thesis, we were able to solve in practice many of the algebraic modellings that we derived from public-key schemes proposed for cryptographic use. This clearly shows that the security level is very low. However, for the systems that we could not solve with our computation capacity, estimating the security is not as simple. The reason is that, for our systems, there exists no efficient way of knowing when the resolution by Gröbner bases is going to terminate before the end of the computation. Ideally, we would like to know the complexity (or simply a bound on the complexity) of the Gröbner bases computation to perform. Such bounds exist for generic systems, but the systems that we have to solve in cryptanalysis are precisely *not* random because they describe a specific mathematical object. For this reason, it would be an important task to study the behavior of the Gröbner bases algorithms with the systems (2) and (3) to determine their complexities. An idea to do this would be to extend the study of the systems of *bi-linear* polynomials (of the form $\sum_{i,j} \alpha_{i,j} X_i Y_j \in \mathbb{K}[X_1, \ldots, X_{v_X}, Y_1, \ldots, Y_{v_Y}]$) performed in [FSS11] to the polynomials $\sum_{i,j} \alpha_{i,j} X_i^{p^u} Y_j^{p^v}$ which are *quasi*-bilinear in characteristic p and appear in the algebraic modellings. Other methods exploiting the fact that the resolution is performed in a finite field could be explored. A first idea is to study the behaviour of the algebraic systems obtained by decomposing each unkown

of a polynomial over $\mathbb{F}_{q^m}$ into m unknown coordinates over $\mathbb{F}_q$. Tighter bounds would allow to give, only with the code parameters, the level of security against algebraic cryptanalysis. Moreover, we explained that the study of algebraic system was a proficient source of ideas to understand the structure of the modeled object. Thus, there is good hope that studying these systems may reveal unexpected properties of the codes, or the existence of a more efficient Gröbner bases computation algorithm dedicated to those algebraic systems.

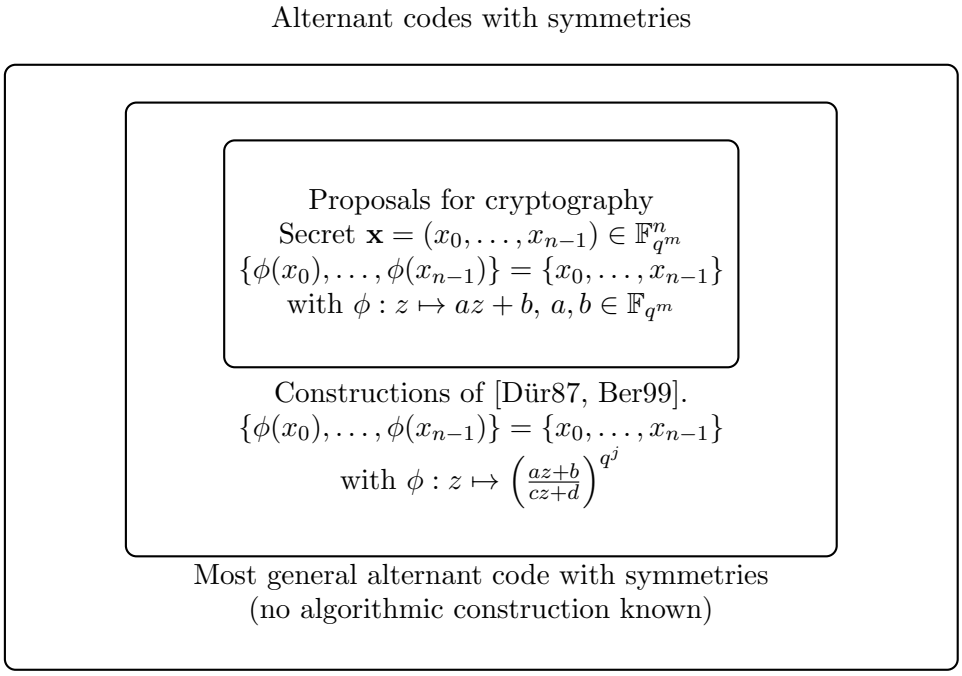
Alternant Codes with Symmetries for McEliece

Knowing if there exist alternant codes with symmetries suitable for cryptography is a natural question. After this thesis, we can split the alternant codes with symmetries into three categories (summed up in the figure on page 15):

1. For the constructions proposed in cryptography so far, which consider alternant codes whose secret element $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_{q^m}^n$ is globally stable by affine maps $\phi : z \mapsto az + b$, we have proved the effectiveness of the folding method. It reduces the key security to that of an alternant code with same key size and *without* symmetry.
2. The construction of alternant codes with symmetries introduced in [Dür87] and [Ber99, Ber00b] and recalled in Chapter 5 is more general. It considers secret elements $\mathbf{x}$ stable by affine maps combined with an action of the Fröbenius endomorphism $\phi : z \mapsto (az + b)^{q^j}$ (for $0 \leqslant j \leqslant m - 1$), and in the most general case maps of the form $\phi : z \mapsto \left(\frac{az+b}{cz+d}\right)^{q^j}$. The similarities with the proposed constructions (algebraic relations on the support) may lead to think that the folding method extends to those cases. This question should receive attention.
3. Symmetric alternant codes out of the scope of Chapter 5 and [Dür87] exist. This means that no map linking the support elements is known. In this case, the folding method is very unlikely to preserve structure. The problem for those codes is that we do not know of any way of generating them efficiently for now.

On the Security of the Original McEliece Cryptosystem

Nowadays, the big key sizes of McEliece encryption are not as crippling as they used to be and the post-quantum feature of code-based cryptography is of growing importance. Amongst code-based primitives, McEliece scheme is the oldest one and it must be emphasized that such longevity has been rarely seen for a public-key cryptosystem and is, for some people, very confidence-inspiring. However, we think that this longevity should not be considered as a security guaranty. Globally, the McEliece cryptosystem itself received little attention for a long time. The attacks put forward between 1980 and 2000 ([LB88, Leo88, Ste88, CC98]) focused on the problem of decoding, that is, solving a linear system with error, which is a very general algorithmic problem. It is only recently, with the SSA algorithm by Sendrier [Sen00], that cryptanalysts started to tackle the question of recovering the secret elements of a Goppa code. The cryptanalytic works that we introduced in this thesis do not threaten its security in practice, but they revealed some unknown properties of Goppa codes, for instance Theorem 7.16 on Goppa codes defined over $\mathbb{F}_{p^s}$ with a polynomial of multiplicity p^s (with p a prime and $s > 1$). Such properties are interesting because they reveal different behaviours for Goppa codes and random codes and allow to distinguish them. Recent improvements of attacks (based on a *code distinguisher* [CGG+14]) represent an exciting axis of research. Thanks to those methods, several schemes using a specific case of alternant code (namely the GRS codes) were broken. GRS codes are alternant codes with one fixed parameter: the



extension degree m is set to 1. It has been thought for long that picking alternant codes with $m > 1$ completely discarded the attacks against GRS codes so that those would not threaten the alternant codes. This belief was proved to be wrong in [COT14], where the authors use a code distinguisher to design a polynomial time attack against Wild Goppa codes with parameter m set to 2 (that is, a sub-family of a sub-family of alternant codes). Extending this work to less specific Goppa and alternant codes is therefore a desirable goal for cryptanalysts.

Résumé

Contexte

La cryptologie étudie les méthodes permettant de protéger des informations secrètes. Historiquement, la cryptologie a été d'une importance majeure pendant les guerres, car la confidentialité des informations échangées entre le centre de commandement et le théâtre des opérations est cruciale pour le succès d'une mission. Un exemple célèbre est celui de Jules César qui utilisait la cryptologie pour protéger sa correspondance. Un autre exemple, plus récent, est fourni par la machine Enigma utilisée par les militaires allemands pendant la Seconde Guerre Mondiale. Il est communément admis que la découverte, par des scientifiques polonais et britanniques, d'un moyen de contourner la protection introduite par Enigma dans les messages échangés aurait raccourci la guerre de deux ans.

De nos jours, l'utilisation de la cryptologie n'est plus cantonnée aux seules communications militaires. Le besoin de transmissions sécurisées concerne chacun d'entre nous, lorsque nous envoyons des courriers électroniques ou lorsque nous payons sur internet. Nous devons également protéger notre identité lorsque nous échangeons avec un serveur à distance. Consciemment ou non, les utilisateurs d'ordiphones [2] enregistrent une quantité importante de données concernant leur vie privée sur un appareil connecté en permanence à des réseaux dont la sécurité est peu claire. Traiter, entre autres, ces différents problèmes, fait partie des objectifs de la cryptologie moderne.

Une étape essentielle pour la cryptologie fut atteinte en 1976 avec l'avènement de la cryptographie *à clef publique*. Avant cette date, lorsque Alice et Bob voulait communiquer de manière sécurisée, il leur était nécessaire d'échanger *au préalable* un secret. Par exemple, César devait transmettre à son interlocuteur un nombre secret pour que celui-ci puisse lire le message protégé. Cet échange secret préalable rend les communications extrêmement coûteuses lorsqu'un grand nombre d'entités souhaite communiquer. La percée réalisée par Diffie et Hellman consista à expliquer comment Alice et Bob peuvent se mettre d'accord sur une valeur secrète et connue d'eux seuls tout en échangeant uniquement des messages lisibles par tous, de telle sorte que l'échange secret préalable n'est plus nécessaire.

La cryptanalyse est le domaine de la cryptologie dédié à l'analyse des méthodes permettant de contourner les protections introduites pour protéger de l'information et retrouver des données vouées à rester secrètes pour une entité non-autorisée. La méthode pour retrouver une information protégée est souvent appelé une *attaque*. Évaluer finement la faisabilité d'une attaque est d'une importance cruciale pour déterminer le niveau de protection apportée à une information. En effet, lorsqu'un appareil utilisant des procédés cryptographiques est vendu, il est nécessaire (et obligatoire) de déterminer son niveau de résistance aux différentes attaques connues. C'est ce que nous faisons dans cette thèse, dans un domaine spécifique de la cryptologie, que nous présentons maintenant.

2. communément appelé smartphone

Les codes correcteurs d'erreurs

Les codes correcteurs d'erreurs sont un outil mathématique dont le but historique est de protéger des données contre les erreurs de transmission. L'idée générale consiste à ajouter de la *redondance* à un message de sorte que, si quelques caractères (ou *bits d'information*) sont altérés lors de la transmission, il soit possible de corriger ces erreurs et retrouver le message intial. La fonction ajoutant la redondance est appelée *l'encodage*, celle corrigeant les erreurs *le décodage*. En pratique, le décodage est possible si le nombre de bits altérés est inférieur à un certain seuil appelé *capacité de correction* de l'encodage.

Les codes correcteurs d'erreurs sont omniprésents dans les communications digitales: les paquets IP sur internet, les messages envoyés par les satellites à travers l'espace sont protégés contre les erreurs de transmission, les données stockées sur un CD sont protégées contre les rayures ... Un grand nombre de fonctions d'encodage a été développé selon les besoins industriels. Par exemple, pour les satellites il est nécessaire d'utiliser un encodage avec une grande capacité de correction. Pour garantir la bonne lecture d'un CD, l'encodage doit permettre un décodage très rapide. De nombreux exemples peuvent être trouvés dans [MS86, PBH98].

Une famille très répandue de codes est la famille des codes *linéaires*, appelés ainsi car l'ensemble des messages encodés forme un espace vectoriel (*i.e* ils forment un sous-espace vectoriel de $\mathbb{F}_q^n$, pour q puissance d'un nombre premier et n un entier appelé *longueur du code*). Ils peuvent être représentés aisément en donnant une base de cet espace vectoriel, par exemple sous la forme d'une matrice. Considérons un message de k bits, nous l'écrirons sous la forme d'un vecteur-ligne $\mathbf{m} = (m_0, \ldots, m_{k-1}) \in \mathbb{F}_2^k$. Un code linéaire admet un encodage simple représenté par $\mathbf{G}$ une matrice $k \times n$, qui peut être choisie de la forme $\mathbf{G} = (\mathbf{I}_k|\mathbf{A})$, où $\mathbf{I}_k$ est la matrice identité de taille $k \times k$, et $\mathbf{A}$ une matrice $k \times (n-k)$. La fonction d'encodage est :

$$\mathbf{m} \in \mathbb{F}_2^k \mapsto \mathbf{mG} = \mathbf{m}(\mathbf{I}_k|\mathbf{A}) = (\mathbf{m}, \mathbf{b}) \in \mathbb{F}_2^n,$$

où $\mathbf{b}$ est la redondance.

En somme, encoder consiste à faire une combinaison linéaire des éléments d'une base du code. Décoder est, en général, plus compliqué. Considérons un message encodé reçu avec des erreurs, nous l'écrivons $\mathbf{m}' = \mathbf{mG} + \mathbf{e}$, où $\mathbf{e}$ est un vecteur de $\mathbb{F}_2^n$ dont toutes les entrées valent 0 sauf aux positions où ont eu lieu des erreurs. Décoder revient à résoudre le système

$$\mathbf{m}' = \mathbf{mG} + \mathbf{e}$$

où $\mathbf{m}', \mathbf{G}$ sont connus et $\mathbf{m}, \mathbf{e}$ inconnus. Ce problème est communément appelé *résoudre un système linéaire avec erreurs*, et fut prouvé NP-difficile (dans [BMvT78, LDW94]).

L'un des buts de la théorie des codes correcteurs linéaires est de trouver des familles spécifiques d'espaces vectoriels pour lesquelles le décodage peut être résolu rapidement (lorsque le nombre d'erreurs est borné).

Le Chapitre 1 de cette thèse donne l'exemple des *codes alternants* et des codes *de Goppa* (binaires ou non), qui sont les codes auxquels nous nous sommes intéressé dans ce travail.

Cryptographie à base de codes correcteurs

Le lien entre codes correcteurs et cryptographie fut établie en 1978 par McEliece dans [McE78]. L'idée est la suivante: pour certaines familles de codes correcteurs pour lesquelles il existe un décodage efficace, il est nécessaire de connaître certains éléments utilisés lors de la construction du code pour appliquer le décodage. Ainsi, un message encodé et altéré devient illisible à toute personne ignorant ces éléments. En termes cryptographiques, la fonction d'encodage est alors une fonction *à sens unique avec trappe*, c'est-à-dire une fonction

difficile à inverser sans la connaissance d'un élément secret. On définit alors un schéma de chiffrement à clef publique de la manière suivante: l'encodage (décrit par une base du code linéaire utilisé, c'est-à-dire une matrice $\mathbf{G}_{pub}$) sert de clef publique, le chiffrement consiste à encoder le message et y introduire des erreurs. La clef privée est l'élement mathématique permettant de décoder efficacement, le déchiffrement se fait précisément en appliquant la procédure de décodage.

$$\overset{\text{plaintext } \mathbf{m}\in\mathbb{F}_q^k}{\begin{pmatrix} 0,\ldots,1 \end{pmatrix}} \overset{\text{public key } \mathbf{G}_{pub}\in\mathbb{F}_q^{k\times n}}{\begin{pmatrix} & & \\ & \mathbf{G}_{pub} & \\ & & \end{pmatrix}} + \overset{\text{error } \mathbf{e}\in\mathbb{F}_q^n}{\begin{pmatrix} 1,0,\ldots,0,1 \end{pmatrix}} = \overset{\text{ciphertext } \mathbf{c}\in\mathbb{F}_q^n}{\begin{pmatrix} 1,0,\ldots,1,1 \end{pmatrix}}$$

Principe général d'un chiffrement à base de codes correcteurs

Trouver des codes avec un élément secret permettant le décodage est une question centrale en cryptographie à base de codes correcteurs. La sécurité du chiffrement repose à la fois sur la difficulté à décoder sans cet élément et sur la difficulté à retrouver cet élément quand la seule fonction d'encodage (publique) est connue. La famille de codes proposée par McEliece (les codes de Goppa binaires), est toujours considérée aujourd'hui comme sûre selon ces deux critères. Dans le Chapitre 2, nous présentons les meilleures méthodes connues pour traiter ces deux problèmes et détaillons leur coût algorithmique.

Après cette première proposition de fonction à sens unique avec trappe, appelée *chiffrement de McEliece*, d'autres primitives cryptographiques utilisant des codes correcteurs ont été introduites. Pour chacune d'entre elle, la sécurité est garantie par la difficulté du problème du décodage d'un code linéaire, et la *cryptographie à base de codes correcteurs* regroupe toutes les primitives partageant cette propriété. Les plus connues sont le *chiffrement de Niederreiter* [Nie86], le *schéma d'authentification de Stern* [Ste93] et le *schéma de signature CFS* (pour Courtois, Finiasz et Sendrier) [CFS01]. Le fait que la sécurité repose sur un problème différent des problèmes de théorie des nombres tels que la factorisation des entiers ou le logarithme discret dans un groupe cyclique est un atout de la cryptographie à base de codes correcteurs. En effet, ces problèmes sont supposés durs pour le moment, mais des progrès algorithmiques ne sont pas à exclure et il est préférable de disposer de primitives cryptographiques dont la sécurité dépende de problèmes aussi variés que possible.

De plus, aucun algorithme quantique n'est connu pour résoudre le problème du décodage d'un code linéaire en temps polynomial, contrairement aux problèmes du logarithme discret et de la factorisation, qui peuvent être résolus en temps $O(\log(n)^3)$ (où $\log(n)$ est la taille de l'entrée) par l'algorithme de Shor [Sho94a, Sho94b] utilisant un ordinateur quantique. De fait, l'émergence d'un ordinateur quantique menacerait la sécurité d'une longue liste de cryptosystèmes à clef publique (l'échange de clef Diffie-Hellman, le chiffrement RSA, la signature ECDSA, la cryptographie utilisant des courbes elliptiques ou hyper-elliptiques ...). Récemment, une telle possibilité a été jugée suffisamment plausible par les organisations de standards cryptographiques pour commencer à préparer la standardisation de primitives post-quantiques, en particuler des cryptosystèmes utilisant des codes correcteurs [3].

3. `http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/Quantum_Safe_Whitepaper_1_0_0.pdf`

Axes de recherche récents en Cryptographie à base de codes correcteurs

Reflétant un intérêt croissant pour la cryptographie à base de codes correcteurs, plusieurs axes de recherche cherchant à rendre les primitives utilisant des codes utilisables en pratique ont été explorés dans les dernières années.

Réduction des coûts de stockage des clefs

Depuis 1978, la cryptographie à base de codes correcteurs n'a pas été déployée à une large échelle. Ceci vient principalement du fait que les clefs à stocker ont une taille importante en comparaison aux autres primitives à clefs publiques (telle que RSA). Par exemple, pour atteindre un niveau de sécurité de 2^{80}, la clef publique requise par le chiffrement de McEliece a une taille soixante fois plus importante qu'une clef RSA. De nombreuses propositions ont été faites pour résoudre ce problème depuis 1985 [Gab85, Nie86, Sid94, JM96, MRS00, Gab05, BL05, BCGO09, MB09, BLM11, BLP11b, BLP11c, Per12a, MTSB13]. Déterminer la sécurité de ces nouvelles primitives est une question cruciale, et de fait beaucoup d'entre elles ont été cassées [SS92, Wie06b, Ove08, FOPT10a, CMCP14].

Dans cette thèse, nous avons étudié deux familles différentes de codes qui remplissent cet objectif de réduction du coût de stockage des clefs. Elles font toutes les deux partie de la grande famille des codes alternants, qui contient aussi les codes de Goppa binaires utilisés par McEliece. Plus précisément, nous nous intéressons à la sécurité des codes de Goppa *symétriques* proposés dans [BCGO09, MB09, BCMN10, BLM11], et des codes de Goppa *sauvages*, introduits dans [BLP11b, BLP11c].

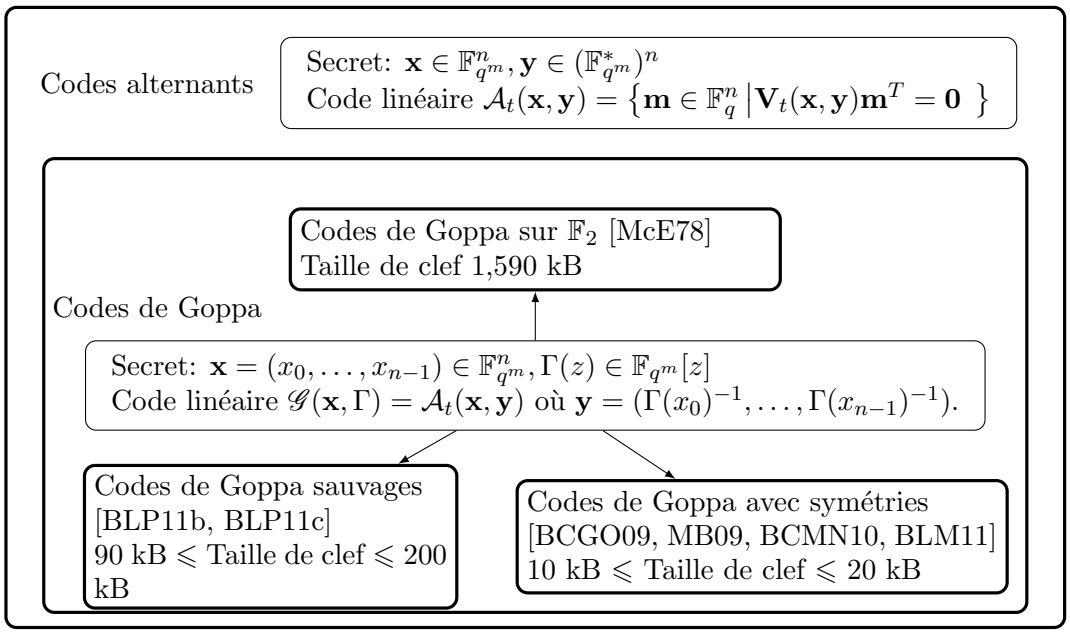
Pour définir ces codes, nous introduisons les matrices de type Vandermonde suivantes:

$$\mathbf{V}_t(\mathbf{x},\mathbf{y}) = \begin{pmatrix} y_0 & \cdots & y_{n-1} \\ y_0x_0 & \cdots & y_{n-1}x_{n-1} \\ \vdots & & \vdots \\ y_0x_0^{t-1} & \cdots & y_{n-1}x_{n-1}^{t-1} \end{pmatrix}, \tag{5}$$

où $\big(\mathbf{x} = (x_0,\ldots,x_{n-1}), \mathbf{y} = (y_0,\ldots,y_{n-1})\big) \in \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n$. Nous résumons dans la figure page 21 les sous-familles des codes alternants que nous étudions dans cette thèse.

Cryptanalyse algébrique en cryptographie à base de codes alternants

La cryptanalyse des primitives utilisant des codes correcteurs a consisté pendant longtemps uniquement en des attaques génériques de recouvrement de message, ce qui revient à résoudre le problème du décodage pour un code linéaire aléatoire. Les meilleures méthodes connues sont dites du type ISD (pour *Information Set Decoding*, ou Décodage par ensemble d'information). Peu de méthodes étaient connues pour retrouver la clef secrète, (à notre connaissance, uniquement [Sen00, LS01]). Récemment, de nouvelles techniques ont été proposées. Elles appartiennent à la catégorie d'attaques appelée *attaques algébriques*. La cryptanalyse algébrique est une méthode d'attaque consistant à lier le secret recherché aux solutions d'un système d'équations algébriques. Pour ce faire, la première étape consiste à trouver une *modélisation algébrique* du secret, c'est-à-dire exprimer le secret comme les inconnues d'un système d'équations algébriques. La seconde étape consiste si possible à résoudre le système, ou à défaut évaluer la complexité de sa résolution. Résoudre un système d'équations non-linéaires est un problème fondamental, communément appelé PoSSo, qui intervient dans de nombreux



Sous-familles des codes alternants considérées dans cette thèse. Les tailles de clefs sont données pour un schéma de chiffrement avec niveau de sécurité 2^{128}.

domaines scientifiques (cryptologie, mécanique, biologie ...). Il fut prouvé NP-difficile lorsque les coefficients des équations et les solutions recherchées appartiennent à un corps fini ([GJ90]). Cependant, en cryptologie, les systèmes que l'on cherche à résoudre sont rarement aléatoires, car ils sont déduits d'objets très structurés provenant d'applications concrètes. Ils ont donc souvent une structure particulière (parfois cachée). Dans de nombreux exemples, les systèmes algrébriques décrivant la clef secrète d'un cryptosystème à clef publique se sont avérés faciles à résoudre en pratique (par exemple dans [FJ03, CM03, FLdVP08, FS10, FOPT10a, BFP11]).

Pour résoudre des systèmes algébriques sur des corps finis, il existe plusieurs méthodes (recherche exhaustive, algorithmes de SAT, ...). Nous utilisons dans cette thèse des méthodes de *bases de Gröbner*, que nous décrivons dans le Chapitre 3. Ces méthodes ont l'avantage de fournir des outils pour analyser la complexité des algorithmes de résolution associés, ce qui est utile pour donner des niveaux de sécurité en cryptographie. De plus, l'analyse du comportement des algorithmes de bases de Gröbner peut se révéler riche en information sur la structure de l'objet modélisé.

Concernant les codes alternants, une modélisation algébrique de la clef secrète a été introduite dans [FOPT10a]. Soit $\mathbf{X} = (X_0, \ldots, X_{n-1})$ et $\mathbf{Y} = (Y_0, \ldots, Y_{n-1})$ des variables dans $\mathbb{F}_{q^m}$, qui correspondent à la clef secrète, et $g_{i,j}$ les entrés de la matrice génératrice $k \times n$ d'un code alternant. Alors, $\mathbf{X}$ et $\mathbf{Y}$ doivent satisfaire:

$$\Big\{ g_{i,0} Y_0 X_0^\ell + \cdots + g_{i,n-1} Y_{n-1} X_{n-1}^\ell = 0 \;\mid\; i \in \{0, \ldots, k-1\}, \ell \in \{0, \ldots, t-1\} \Big\}. \tag{6}$$

En pratique, pour des paramètres cryptographiques, cette modélisation ne permet pas de retrouver la clef secrète d'un schéma McEliece car le coût de résolution du système (6) est trop élevé pour les capacités actuelles de calcul. Cependant, cette modélisation a été utilisée dans [FGO+13] pour construire un algorithme de complexité polynomiale capable de

déterminer si une matrice donnée est la matrice génératrice d'un code alternant (lorsque celui-ci a une grande dimension). Il était conjecturé jusqu'alors (dans [Sen02]) qu'il n'existait pas d'algorithme efficace pour répondre à ce problème (appelé *problème du distingueur*). De plus, grâce à cette modélisation, une attaque en recouvrement de clef a été mise au point sur une famille particulière de codes alternants avec symétries. Cependant, cette attaque n'était pas applicable pour les codes définis sur un corps premier. Ceci a donc été utilisé pour proposer des codes avec symétries résistants aux attaques algébriques [BCMN10, BLM11].

Conception d'implémentations sûres et efficaces

Une étape importante pour utiliser des cryptosystèmes utilisant des codes dans des applications pratiques consiste à trouver des implémentations sûres. En effet, l'implémentation d'une primitive cryptographique peut introduire des vulnérabilités face aux attaques dites *par canaux auxiliaires*.

Les attaques par canaux auxiliaires reposent sur une méthode permettant de retrouver de l'information secrète introduite par Kocher dans [Koc96]. Dans cet article, Kocher montre qu'il est possible de retrouver une clef secrète RSA en mesurant le temps d'éxecution du déchiffrement. L'idée centrale est d'exploiter le fait que le nombre d'opérations (et donc le temps d'exécution) du déchiffrement dépend de la valeur des bits de la clef secrète. L'attaque consiste à faire une hypothèse sur la valeur des bits de clef secrète et vérifier la conformité du temps d'exécution avec la prédiction. Depuis, d'autres mesures physiques ont été utilisées pour monter des attaques (par exemple, la consommation électrique [KJJ99, ÖOP03], le rayonnement électro-magnétique [GMO01] et les ondes acoustiques [GST14]). L'analyse des attaques par canaux auxiliaires est devenue un domaine de recherche à part entière, et il est devenu impératif d'implémenter les algorithmes manipulant des données secrètes de manière à rendre ces attaques impossibles. Pour ce faire, les émanations physiques doivent être rendues indépendantes des secrets manipulés. Si ce n'est pas le cas, on dit qu'il y a une *fuite*, et le but des attaques physiques est de mesurer et exploiter ces fuites. Les modifications apportées à une implémentation pour éliminer ces fuites sont appelées des *contre-mesures*.

Trouver des contre-mesures est un travail spécifiques à chaque opération mathématique (par exemple, l'exponentiation d'un élément dans un groupe cyclique ou sur une courbe elliptique, l'addition de deux vecteurs, le produit matrice-vecteur ...). Il existe une littérature riche sur ce sujet pour les opérations intervenant dans les primitives les plus répandues telles que RSA, mais pour les primitives utilisant des codes correcteurs ce n'est que récemment que les chercheurs ont commencé à étudier les fuites présentes dans une implémentation trop naïve (par exemple dans [SSMS09, AHPT11, Str10b, Str13]) et à proposer des implémentations espérées plus résistantes [Hey11, Str10a, SWM$^+$10, BCS13].

Certains articles ont proposé des implémentations complètes de cryptosystèmes à base de codes, par exemple [CGP08, Str10a, Bis10, SWM$^+$10, Hey11, LS12, BCS13]. Certains prennent en compte les attaques par canaux auxiliaires, d'autres se concentrent sur l'efficacité. En parallèle, d'autres travaux d'attaque sur des implémentations du chiffrement McEliece ont été menés [SSMS09, AHPT11, Str10b, Str13]. Ces articles identifient des fuites et proposent la plupart du temps des contre-mesures pour les éliminer. Cependant, l'efficacité de ces contre-mesures n'est pas totale et l'attaque proposée dans [Str13] est toujours possible.

Garantir la conformité avec des modèles de sécurité

Les modèles de sécurité sont des abstractions des primitives cryptographiques qui permettent de mesurer et borner la quantité d'information qu'un attaquant peut espérer retrouver. Ce sont des abstractions car elles ne considèrent par les propriétés mathématiques des

primitives considérées mais seulement des aspects cryptographique. Par exemple, pour une fonction de chiffrement, les modèles de sécurité mesurent la capacité d'un attaquant à lier un message chiffré avec le message clair correspondant. Au contraire, pour prouver qu'une fonction cryptographique satisfait un modèle de sécurité, il est nécessaire d'examiner ses propriétés mathématiques. En cryptographie à base de codes correcteurs, plusieurs travaux se sont attachés à évaluer les exigences de sécurité satisfaites par les primitives courantes, et comment les modifier pour qu'elles satisfassent des modèles de sécurité, par exemple dans [Ber97, KI01, NIKM08, NC11, Per12b, DDMQN12].

Contributions

Pour les propositions récentes de codes linéaires avec clef compacts (codes alternants avec symétrie et codes de Goppa sauvages), nous révélons des propriétés inconnues qui se sont avérées être des faiblesses. En effet, grâce à ces dernières, nous introduisons de nouvelles attaques de recouvrement de clef qui permettent de retrouver en pratique la clef privée pour de nombreux paramètres proposés pour un usage cryptographique. Ces nouvelles attaques devront être prises en compte à l'avenir pour proposer des paramètres sûrs. Elles appartiennent à la catégorie des attaques algébriques.

Cryptanalyse algébrique dans cryptographie à base de codes alternants

Dans le Chapitre 4, nous étudions une attaque algébrique contre tous les codes alternants déduite de la modélisation (6). Nous améliorons cette modélisation en décrivant plus finement certaines sous-familles des codes alternants. Nous introduisons de nouvelles modélisations, où les blocs de variables $\mathbf{X}$ et $\mathbf{Y}$ du système (6), qui contiennent n variables chacun, sont remplacés par deux blocs contenant chacun $n-k$ variables. Nous donnons des bornes de complexité pour la résolution des systèmes algébriques associés. Pour ce faire, nous supposons que, par recherche exhaustive, suffisamment de variables ont été retrouvées pour pouvoir déduire toutes les autres variables en résolvant un système linéaire. Grâce à cette méthode, nous montrons que le nombre d'opérations nécessaires pour retrouver la description secrète d'un code alternant est borné supérieurement par $q^{m^2 t} \cdot \mathrm{poly}(n)$. Nous comparons ce résultat avec la complexité de l'autre attaque sur la clef connue pour les codes de Goppa (l'attaque par séparation du support). Nous montrons que, asymptotiquement, l'attaque algébrique est plus efficace pour attaquer les codes dont la longueur est significativement plus petite que la taille du corps où les éléments secrets sont choisis (*i.e.* $n \ll q^m$).

Schéma McEliece utilisant des codes alternants avec symétries

Les codes alternants avec symétries sont des codes alternants dont une matrice génératrice est faite de blocs symétriques. Une telle matrice peut être décrite entièrement en ne stockant qu'un nombre réduit de ses entrées. Sur la figure page 24, stocker 16 éléments a_i est suffisant pour connaître entièrement une matrice contenant 64 entrées. Des codes de ce type ont été introduits par différents auteurs [BCGO09, MB09, BCMN10, BLM11]. Leur but est de concevoir un schéma de chiffrement dont la clef publique a un coût de stockage réduit. Par exemple, les cryptosystèmes proposés dans [MB09] ont des tailles de clef réduite d'un facteur entre 22 et 117 en comparaison à un code de Goppa binaire de même sécurité. Faugère, Otmani, Perret et Tillich (FOPT) ont montré que la modélisation des codes alternants (6) peut être adaptée à ces codes avec symétries, mais le système obtenu ne pouvait pas être résolu dans tous les cas. Des attaques, des attaques pratiques étaient rendues possibles pour

tous les paramètres proposés dans [BCGO09] et pour certains provenant de [MB09], mais pour aucun de ceux de [BCMN10, BLM11].

Dans le Chapitre 5, nous rassemblons tous les codes alternants avec symétries autour d'une même propriété (précisément, ils ont tous un groupe d'automorphismes non-trivial). Grâce à cette propriété, nous unifions leur construction et expliquons comment, à partir d'un code alternant usuel, construire un code alternant avec symétrie. Par exemple, nous montrons que tous les codes proposés dans [BCGO09, MB09, BCMN10, BLM11] peuvent être obtenus en choisissant des supports $\mathbf{x} \in \mathbb{F}_{q^m}^n$ globalement stables par une fonction affine $\phi : z \mapsto az + b$ (où $(a, b) \in \mathbb{F}_{q^m}^* \times \mathbb{F}_{q^m}$). Puis, nous montrons que, pour ces codes, il est possible d' *inverser* cette construction. Pour ce faire, nous contruisons, à partir d'une matrice génératrice avec des blocs symétriques, une matrice où chaque bloc a été remplacé par la somme des coordonnées du bloc. Nous appelons cette construction le *pliage* (ou *folding*).

$$\left(\begin{array}{cccc|cccc}
a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\
a_1 & a_0 & a_3 & a_2 & a_5 & a_4 & a_7 & a_6 \\
a_2 & a_3 & a_0 & a_1 & a_6 & a_7 & a_4 & a_5 \\
a_3 & a_2 & a_1 & a_0 & a_7 & a_6 & a_5 & a_4 \\
\hline
a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\
a_9 & a_8 & a_{11} & a_{10} & a_{13} & a_{12} & a_{15} & a_{14} \\
a_{10} & a_{11} & a_8 & a_9 & a_{14} & a_{15} & a_{12} & a_{13} \\
a_{11} & a_{10} & a_9 & a_8 & a_{15} & a_{14} & a_{13} & a_{12}
\end{array}\right) \xrightarrow{\text{Pliage}} \left(\begin{array}{c|c}
a_0 + a_1 & a_4 + a_5 \\
+a_2 + a_3 & +a_6 + a_7 \\
\hline
a_8 + a_9 & a_{12} + a_{13} \\
+a_{10} + a_{11} & +a_{14} + a_{15}
\end{array}\right)$$

Pliage d'une matrice 8×8 avec symétries, faite de blocs symétriques 4×4, en une matrice 2×2.

La faiblesse des codes alternants avec symétries vient du fait que ce code plié est un code alternant dont les éléments privés sont très proches de ceux du code de départ. Nous prouvons ceci en montrant que, si ϕ est la fonction affine utilisée pour construire le support du code de départ et $\alpha \in \mathbb{F}_{q^m}^*$, il existe des polynômes $A_{\phi,\alpha}, B_{\phi,\alpha} \in \mathbb{F}_{q^m}[z]$ tels que tout polynôme $P \in \mathbb{F}_{q^m}[z]$ vérifiant $P(\phi(z)) = \alpha P(z)$ peut être écrit sous la forme $P(z) = A_{\phi,\alpha}(z)Q(B_{\phi,\alpha}(z))$, où $Q \in \mathbb{F}_{q^m}[z]$ est de degré strictement inférieur à celui de P. Ceci mène à notre contribution centrale sur les codes alternants avec symétries, résumée ci-dessous.

Theorem (informel, voir Théorème 5.21 page 98)**.** *Soit $\mathscr{C}$ un code alternant de longueur n, avec pour secret $(\mathbf{x}, \mathbf{y}) \in \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n$, dont une matrice génératrice est faite de blocs symétriques de taille $t \times t$ comme dans [BCGO09, MB09, BCMN10, BLM11]. Alors, le code* plié *de $\mathscr{C}$ est un code alternant $\mathscr{C}'$ de longueur n/t et de secret $(\mathbf{x}', \mathbf{y}') \in \mathbb{F}_{q^m}^{n/t} \times \mathbb{F}_{q^m}^{n/t}$. De plus, il existe une relation explicite entre $(\mathbf{x}', \mathbf{y}')$ et $(\mathbf{x}, \mathbf{y})$ telle que le secret $(\mathbf{x}, \mathbf{y})$ définissant $\mathscr{C}$ peut être retrouvé efficacement en connaissant $(\mathbf{x}', \mathbf{y}')$.*

La clef secrète d'un code alternant avec symétrie peut donc être retrouvée à partir de celle du code plié, qui lui n'admet plus de symétrie. Ceci prouve que les schémas McEliece utilisant ces clefs compactes ont la même sécurité (pour la clef) qu'un schéma McEliece avec une clef beaucoup plus petite.

Dans le Chapitre 6, nous combinons cette faiblesse des codes alternants avec symétrie avec les nouvelles modélisations algébriques introduites au Chapitre 4. Nous obtenons une attaque algébrique contre les schémas McEliece utilisant des codes alternants (ou de Goppa) dont les supports sont stables sous l'action de fonctions de la forme $\phi : z \mapsto z + b$ (avec $b \in \mathbb{F}_{q^m}^*$), qui sont ceux utilisés dans [MB09, BCMN10, BLM11]. Nous nous sommes restreints à ce cas, car les autres fonctions affines possibles ($\phi : z \mapsto az + b$ avec $a \neq 1$), utilisées dans [BCGO09],

avaient déjà été prouvées très faibles dans [FOPT10a]. Au contraire, cette dernière attaque était structurellement inefficace pour les paramères de [BCMN10, BLM11]. Les résultats pratique que nous obtenons montrent que l'attaque est fatale aux schémas utilisant des codes de grande dimension k (*i.e.* proche de la longueur n), qui sont toujours ceux utilisés pour des applications de signature. Concernant les schémas de chiffrement, l'attaque est de moins en moins efficace à mesure que la dimension du code utilisé diminue. Cependant, les codes de petite dimension sont plus vulnérables aux attaques sur les messages de type ISD. Pour cette raison, il est improbable que des paramètres de chiffrement sûrs puissent être trouvés.

Paramètres (q, m, t, n)	Cette thèse	Attaque précédente	Niveau de sécurité	Ref
$(2^4, 4, 128, 4096)$	0.010 s.	7.1 s	128	[MB09]
$(2^2, 8, 64, 3584)$	0.040 s.	1,776.3 s	128	[MB09]
$(2, 16, 32, 4864)$	18 s.	N.A.	128	[MB09]
$(2, 14, 13, 16368)$	3.0 s	N.A	81	[BCMN10]
$(2, 14, 14, 16368)$	3.0s	N.A.	84	[BCMN10]
$(2, 15, 12, 32752)$	5.4s	N.A	82	[BCMN10]
$(3, 11, 9, 177048)$	3.4 s	N.A.	80	[BLM11]
$(5, 8, 15, 390495)$	82 s	N.A.	128	[BLM11]
$(241, 3, 241, 964)$	0.020 s	N.A.	112	[BLM11]

Temps de recouvrement de la clef privée pour des codes alternants avec symétrie (N. A. signifie Non-Applicable).

Cryptanalyse du schéma McEliece utilisant des codes de Goppa sauvages *(Wild McEliece)*

Le Chapitre 7 est consacré aux schémas *Wild McEliece* et *Wild McEliece Incognito*. Ce sont des schémas de chiffrement à base de codes décrits par Bernstein, Lange et Peters dans [BLP11b] et [BLP11c]. La fonction de chiffrement est la même, mais les codes linéaires utilisés sont décrits par des matrices plus petites que pour McEliece, ce qui confère une taille moindre à la clef publique et un coût de stockage réduit d'un facteur allant jusqu'à 16. Ces codes, qui appartiennent à la famille des codes alternants, peuvent être vus comme une généralisation des codes de Goppa sur $\mathbb{F}_2$ (proposés par McEliece) à d'autres corps finis $\mathbb{F}_q$ avec $q >$. Ils ont été appelés *codes de Goppa sauvages* (ou *wild Goppa codes*) par les concepteurs de ces schémas.

Nos résultats montrent que, grâce à des propriétés spécifiques aux codes de Goppa sauvages, la clef secrète peut être modélisée par un système d'équations contenant un nombre de variables très faible comparé aux systèmes de type (6) décrivant des codes alternants génériques. Soit $g_{i,j}$ les entrées d'une matrice génératrice $k \times n$ d'un code de Goppa sauvage, et $\mathbf{Z} = (Z_0, \ldots, Z_{n-1})$ des variables dans $\mathbb{F}_{q^m}$, nous montrons qu'un attaquant capable de résoudre le système

$$\Big\{ g_{i,0} Z_0^\ell + \cdots + g_{i,n-1} Z_{n-1}^\ell = 0 \ \mid \ i \in \{0, \ldots, k-1\}, \ell \in \{1, \ldots, q-1\} \Big\}, \tag{7}$$

est capable de retrouver la clef secrète.

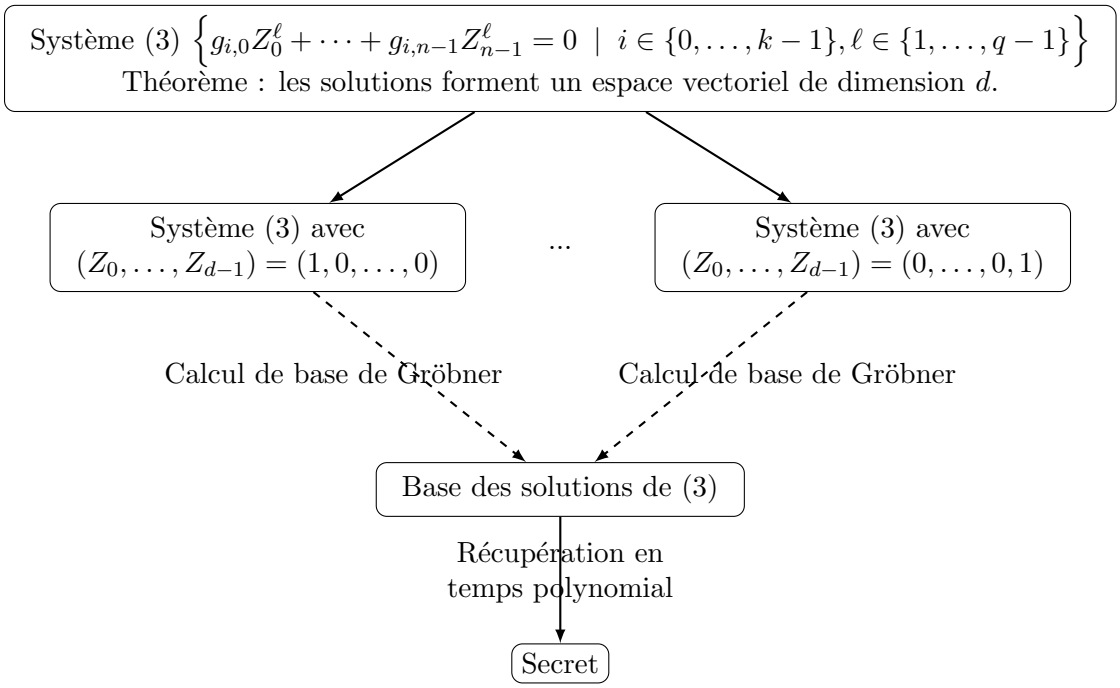
Nous montrons ceci en prouvant que les solutions du système (7) forment un espace vectoriel, ce qui est inhabituel pour un système algébrique non-linéaire. Nous exploitons cette

propriété dans la phase de résolution : comme nous ne cherchons pas un vecteur en particulier mais simplement une base de l'espace des solutions, nous pouvons fixer arbitrairement des coordonnées du vecteur recherché. Une fois une base de l'espace des solutions trouvée, nous concevons une méthode pour retrouver la clef secrète à partir de cette base en temps (heuristiquement) polynomial. Nous résumons dans le théorème suivant l'essentiel de notre résultat.

Theorem (informel, voir les Théorèmes 7.6 (page 122) et 7.15 (page 129))**.** *Soit $\mathscr{C}$ un code Goppa sauvage sur $\mathbb{F}_q$ et $g_{i,j}$ les entrées d'une matrice génératrice $k \times n$. Alors,*

1. *les solutions du système* (7) *forment une union d'espaces vectoriels.*
2. *Étant donnée une base de l'un de ces espaces vectoriels, une description secrète de $\mathscr{C}$ peut être retrouvée en temps (heuristiquement) polynomial.*

Nous en déduisons la stratégie (simplifiée) de cryptanalyse suivante.



Stratégie de recouvrement de clef pour Wild McEliece

Dans un contexte plus large, notre modélisation algébrique avec faible nombre de variables peut être appliqué dès que le polynôme secret $\Gamma(z)$ admet un facteur avec multiplicité. Ceci devrait encourager les concepteurs à choisir des polynômes de Goppa irréductibles.

Nous montons une attaque pratique utilisant cette stratégie contre les codes de Goppa sauvages définis sur des corps non-premiers, sur lesquels nous exhibons des faiblesses supplémentaires. En effet, nous montrons que, pour un code de Goppa sauvage défini sur $\mathbb{F}_{p^s}$ avec p premier et $s > 1$, il est possible de remplacer le système (7) (contenant $(p^s-1)mt-t$ variables) par une variante contenant moins de $(p-1)mst$ variables, puis appliquer la même stratégie de recouvrement de clef. Ceci affecte gravement la sécurité de nombreuses instances proposées pour un usage cryptographique avec un niveau de sécurité de 2^{128} par les auteurs des articles [BLP11b, BLP11c]. D'autres instances ont été proposées sur un site internet comme défis

pour la communauté. Pour plusieurs tailles de corps, nous pouvons retrouver la clef privée pour les instances mises en ligne les plus difficiles.

Paramètres (q, m, t, u, n)	Taille de clef (kB)	Sécurité annoncée	Modélisation générique		**Notre modélisation**	
			Var.	Temps de rés.	Var.	Temps de rés.
$(32, 2, 3, 24, 852)$	90	2^{130}	462	∞	**18**	0.6s
$(27, 3, 2, 42, 1500)$	204	2^{128}	564	∞	**26**	0.9s
$(27, 3, 5, 0, 1700)$	304	2^{158}	780	∞	**65**	1h 59min
$(25, 3, 3, 25, 1206)$	55	2^{117}	582	∞	**57**	1h 2min
$(16, 3, 6, 16, 1328)$	160	2^{125}	636	∞	**54**	36h 35min
$(9, 3, 6, 14, 728)$	40	2^{81}	372	∞	**54**	25h 13min

Nombre de variables et temps de résolution (avec l'implémentation Magma [BCP97a] de F_4) de notre modélisation algébrique pour des codes de Goppa sauvages définis sur $\mathbb{F}_q$ avec polynôme secret $\Gamma(z) = f(z)g(z)^{q-1} \in \mathbb{F}_{q^m}[z]$ où $t = \deg(g)$ et $u = \deg(f)$.

Conclusion sur la cryptanalyse algébrique des codes de Goppa

Tout au long de cette thèse, nous avons conçu des cryptanalyses exploitant des caractéristiques de codes alternant (ou de Goppa) non-génériques. Il est important de souligner qu'aucune de ces attaques ne s'appliquent à des alternant aléatoires. Pour résumer, nous avons exploité les propriétés non-générique suivantes :

— l'existence de relations algébriques vérifiées par les élements du support et les multiplieurs d'un code alternant,
— la non-irréductibilité du polynôme d'un code de Goppa liée à la présence d'un facteur multiple (sauf pour les codes binaires)
— l'appartenance des entrées de la matrice publique à un code non-premier (à l'exception là encore de $\mathbb{F}_2$).

Les codes de Goppa binaires dont le polynôme secret est irréductible n'ont aucune de ces propriétés. Il est intéressant de noter que chaque modification apportée à ces codes s'est avérée être une faiblesse, ce qui devrait encourager les concepteurs à se limiter à ces codes (au sein de la famille des codes de Goppa ou alternants).

Nos résultats illustrent également comment l'étude du comportement des modélisations algébriques peut révéler des propriétés inattendues des objets modélisés.

Prenons l'exemple des codes alternants avec symétries. [FOPT10a] fournit une modélisation algébrique adaptés aux codes alternants symétriques obtenus en considérant les fonctions affines de la forme $\phi : z \mapsto z + b$ (avec $b \in \mathbb{F}_{q^m}^*$). Le système obtenu est très sur-déterminé, *i.e.* il contient bien plus qu'équations que d'inconnues. Pour un système sur-déterminé aléatoire, la probabilité d'avoir des solutions est très faible. Comme le système considéré ici décrit une clef secrète correspondant au code public, le système a nécessairement des solutions (mais très peu). Cependant, nous avons observé le fait suivant : une fois fixé un petit sous-ensemble de variables à la valeur 0, le système obtenu a encore des solutions. Plus étrange encore, pour un autre sous-ensemble de variable, les solutions sont les mêmes que le premier sous-ensemble ait été fixé à 0 ou non. Cette propriété, inattendue et intrigante, s'explique par la structure du code plié : fixer certaines variables à 0 transforme la modélisation algébrique du code de départ en celle de son code plié, qui a donc nécessairement des solutions.

Notre travail sur Wild McEliece fournit un autre exemple de structure révélée par l'étude des systèmes algébriques. En effet, il est possible de prouver que, si les $g_{i,j}$ sont les entrées

d'un code de Goppa sauvage, et $\mathbf{Y}$ est défini comme dans le système (6), on peut écrire un système très surdéterminé d'équations *linéaires*. En effet, lorsque les entrée $g_{i,j}$ du code sont choises dans $\mathbb{F}_q$ avec $q = p^s$, alors les multiplieurs du code sont solutions du système d'inconnues $\mathbf{Y}$:

$$\left\{ g_{i,0}^{p^u} Y_0 + \cdots + g_{i,n-1}^{p^u} Y_{n-1} = 0 \ \mid \ i \in \{0, \ldots, k-1\}, 0 \leqslant u \leqslant s-1 \right\}. \tag{8}$$

Si le système (8) est de rang plein, alors la situation est très favorable pour un attaquant. Le secret peut être retrouvé en temps polynomial. Cependant, nous avons observé au contraire que ces équations linéaires avaient toujours au défaut de rang d'une valeur très spécifique (précisément $n-(p-1)mst$ avec $q = p^s$). Cette chute de rang a pu être reliée avec la dimension d'un code de Goppa sauvage en lien avec le code public, et nous avons déduit une résultat surprenant sur la structure du code public (Théorème 7.16).

Contributions dans la sécurisation des implémentations du cryptosystème McEliece

Notre contribution pour la sécurisation des implémentations des primitives cryptographiques à base de codes correcteurs consiste à proposer une manière d'implémenter le déchiffrement du cryptosystème de McEliece plus résistante aux attaques par canaux auxiliaires que les versions précédentes. Dans le Chapitre 8, nous montrons que l'attaque décrite dans [Str13] peut être améliorée, de sorte qu'il est impératif d'en proposer une contre-mesure fiable. Puis, nous décrivons une telle contre-mesure. Pour ce faire, nous détaillons une version de l'algorithme d'Euclide étendu qui, une fois les paramètres publics fixés, requiert un nombre fixe de boucles, chaque boucle contenant un nombre fixe d'opérations. Ainsi, les attaques par mesure du temps d'exécution sont écartées. Nous apportons donc une brique important dans la construction d'une implémentation sûre du chiffrement McEliece, et plus généralement de toute primitive nécessitant d'implémenter le décodage d'un code alternant.

Perspectives

Analyse des modélisations algébriques

Dans cette thèse, nous avons pu résoudre en pratique beaucoup des modélisations algébriques introduites pour décrire la clef privée de schémas cryptographiques, ce qui met en évidence un faible niveau de sécurité. Pour les systèmes que nous n'avons pas pu résoudre avec notre capacité de calcul, estimer la sécurité n'est pas aussi simple. Ceci vient du fait que, pour les systèmes considérés, il n'existe pas de moyen efficace de savoir quand la résolution par bases de Gröbner se terminera avant la fin du calcul. Idéalement, nous voudrions pouvoir donner la complexité (ou simplement une borne sur la complexité) du calcul de base de Gröbner à résoudre. De telles bornes existent pour des systèmes tirés au hasard, mais ceux que nous avons à résoudre en cryptanalyse ne sont précisément pas tirés au hasard car ils décrivent des objets mathématiques structurés. Pour cette raison, il serait souhaitable d'étudier spécifiquement le comportement des algorithmes de base de Gröbner avec les systèles (6) et (7) afin d'en déterminer la complexité. Une idée pour faire ceci serait d'étendre l'étude des systèmes *bilinéaires* (c'est-à-dire de la forme $\sum_{i,j} \alpha_{i,j} X_i Y_j \in \mathbb{K}[X_1, \ldots, X_{v_X}, Y_1, \ldots, Y_{v_Y}]$) réalisée dans [FSS11] aux polynômes $\sum_{i,j} \alpha_{i,j} X_i^{p^u} Y_j^{p^v}$ qui sont *quasi*-bilinéaires en caractéristique p et qui apparaissent dans les modélisations algébriques. Une autre méthode serait d'exploiter le fait

que la résolution est opérée dans un corps fini. Par exemple, il serait intéressant d'étudier le comportement des systèmes obtenus en décomposant chaque inconnue sur $\mathbb{F}_{q^m}$ en un vecteur de m coordonnées sur $\mathbb{F}_q$. Des bornes plus fines permettraient de donner, à partir des seuls paramètres d'un code, le niveau de sécurité contre les cryptanalyses algébriques connues. De plus, comme nous l'avons expliqué, l'étude des systèmes algébriques peut révéler des propriétés inattendues sur le code, ou l'existence d'algorithmes de calculs de bases de Gröbner dédiés à ces systèmes.

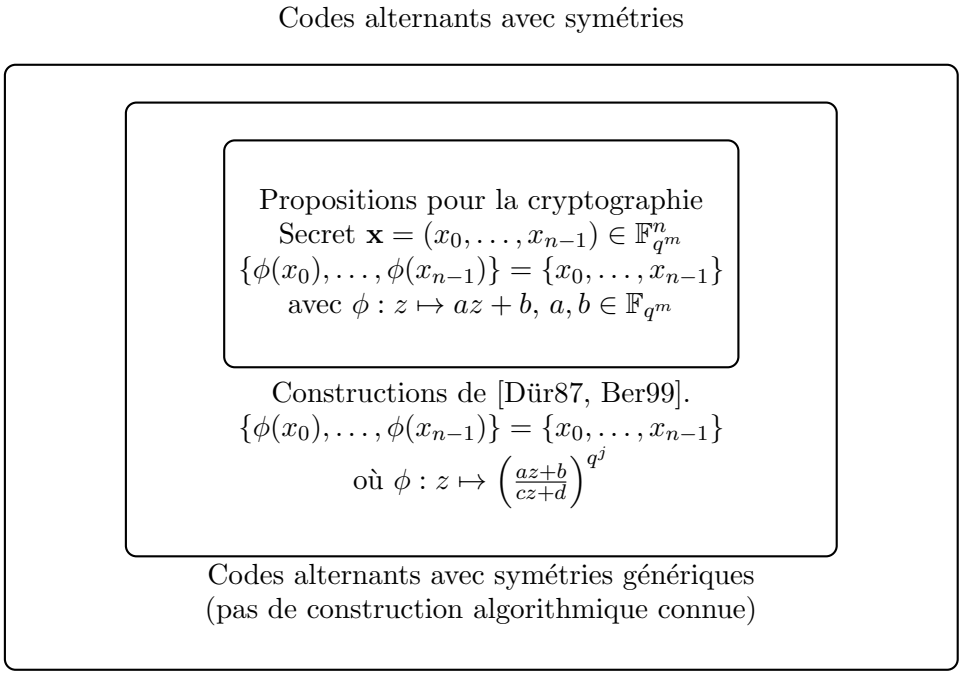
Codes alternants symétriques pour McEliece

Il est naturel de se demander si il existe des codes alternants symétriques garantissant un bon niveau de sécurité pour le schéma McEliece correspondant. Après cette thèse, on peut séparer les codes alternants avec symétries en trois catégories (résumées sur la figure page 30):

1. Pour les constructions proposées jusqu'à présent pour utilisation cryptographique, qui utilisent des codes alternants dont le support secret $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_{q^m}^n$ est globalement stable par une fonction affine $\phi : z \mapsto az+b$, nous avons prouvé l'efficacité de la méthode du pliage. Elle réduit la sécurité de la clef à celle d'un code sans symétrie et beaucoup plus petit.
2. Les constructions de codes alternants avec symétries introduites par Dür et Berger dans [Dür87] et [Ber99, Ber00b], rappelées au Chapitre 5, sont plus générales. Elles utilisent des codes dont l'élement secret $\mathbf{x}$ est stable par une fonction affine composée avec l'action d'un morphisme de Fröbenius $\phi : z \mapsto (az+b)^{q^j}$ (pour $0 \leqslant j \leqslant m-1$), et même dans le cas le plus général par des fonctions de la forme $\phi : z \mapsto \left(\frac{az+b}{cz+d}\right)^{q^j}$. La similarité avec les constructions précédentes (*i.e* les relations algébriques imposées sur le support) laisse à penser que la méthode du pliage peut être étendue à ces codes. Cette question mérite beaucoup d'attention.
3. Des codes alternants symétriques hors de la portée du Chapitre 5 existent. Ceci signifie qu'il n'existe pas de relations algébriques spécifiques sur les éléments secrets (mais qu'il existe malgré tout des automorphismes non-triviaux). Dans ce cas, la méthode du pliage ne semble pas pouvoir s'étendre. Cependant, nous ne connaissons aucun moyen de générer efficacement de tels codes.

Conclusion sur la sécurité du cryptosystème McEliece

Aujourd'hui, les tailles de clef importants sur chiffrement McElice ne sont plus aussi handicapantes qu'elles ne l'étaient dans les années 80. De plus, le caractère post-quantique de la cryptographie à base de codes correcteurs est un atout de plus en plus important. Parmi les primitives utilisant des codes correcteurs, le schéma de McEliece est le plus ancien et une telle longévité est rare pour un cryptosystème à clef publique. Ceci est, pour certains, garant de sa sécurité. Cependant, nous pensons que sa longévité ne devrait pas être considérée ainsi. De manière générale, le chiffrement McEliece en lui-même a été assez peu étudié en lui-même pendant de nombreuses années. Les attaques introduites entre 1980 et 2000 ([LB88, Leo88, Ste88, CC98]) traitent essentiellement du problème du décodage d'un code linéaire alétoire, c'est-à-dire résoudre un système liénaire avec erreurs, qui est un problème algorithmique très général. Ce n'est que récemment, avec l'algorithme SSA de Sendrier [Sen00], que le cryptanalystes ont étudié la difficulté de retrouver la description secrète d'un code de Goppa.



Le travail de cryptanalyse que nous avons mené dans cette thèse ne menace sa sécurité en pratique, mais révèle des propriétés inconnues des codes de Goppa inconnues jusqu'alors (par exemple le Théorème 7.16 traitant des codes de Goppa définis sur $\mathbb{F}_{p^s}$ avec un polynôme secret de multiplicité p^s, où p est premier et $s > 1$). De telles propriétés sont particulièrement intéressantes car elles révèlent des différences de comportement entre les codes de Goppa et les codes aléatoires, et permettent donc de les distinguer. Des améliorations récentes sur des attaques utilisant un *distingueur de code* ([CGG+14]) représentent un axe de recherche prometteur. Grâce à ces méthodes, plusieurs schémas utilisant une famille particulière de codes alternants (les codes GRS) ont été cassées. Les codes GRS sont des codes alternants où un paramètre a été fixé : le degré d'extension m est fixé à 1. Pendant longtemps, il était considéré que les codes alternants pour lesquels m est choisi strictement supérieur à 1 étaient complètement hors de portée des attaques utilisant un distingueur de code. Ceci s'est avéré faux, comme il a été montré très récemment dans [COT14]. Les auteurs utilisent un distingueur de code pour monter une attaque à la complexité polynomiale contre des codes de Goppa sauvages dont le degré d'extension m vaut 2 (c'est-à-dire contre une sous-famille d'une sous-famille des codes alternants). Étendre ce travail à des codes de Goppa ou alternants moins spécifiques pourrait sérieusement affecter la confiance dans ces codes.

Part I

Preliminaries

Chapter 1

Coding Theory

1.1 Code basics

We introduce the definitions and basic tools that will be of constant use in this thesis. We focus only on the aspects of coding theory useful for code-based cryptography. In particular, we insist on the fact that we will use only *linear* codes even though we will sometimes omit the *linear*.

1.1.1 First definitions

Firstly, we give the classical ways to describe a linear error-correcting code. The following definitions are extracted from [MS86][Ch. 1].

Definition 1.1 (Linear Code)**.** *Let $\mathbb{F}_q$ be a finite field of $q = p^s$ elements (p prime, and $s > 0$), n and k be non-negative integers with $k \leqslant n$. A* linear code *$\mathscr{C}$ of* length *n and* dimension *k over $\mathbb{F}_q$ is a subspace of dimension k of the full space $\mathbb{F}_q^n$. It can be specified by a full-rank matrix called a* generator matrix *which is a $k \times n$ matrix $\mathbf{G}$ over $\mathbb{F}_q$ whose rows form a basis of the code, namely*

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

It can also be defined as the right kernel of an $(n-k) \times n$ matrix $\mathbf{H}$ over $\mathbb{F}_q$, called a parity-check *matrix, that is*

$$\mathscr{C} = \left\{ \mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{H}\mathbf{x}^T = \mathbf{0}_{n-k} \right\},$$

where $\mathbf{x}^T$ is the transpose of the row vector $\mathbf{x}$. Therefore, it holds that

$$\mathbf{H}\mathbf{G}^T = \mathbf{0}_{(n-k)\times k}.$$

A vector of a linear code $\mathscr{C}$ is called a codeword *of $\mathscr{C}$. We refer to the quotient $R = k/n$ as the* information rate *of the code.*

To describe linear codes, we will use alternatively generator matrices or parity-check matrices. It is not hard to switch from one description to the other one, thanks to the following classical lemma, that we will use extensively :

Lemma 1.2. *Let $\mathscr{C}$ be linear code of* length *n and* dimension *k over $\mathbb{F}_q$. Suppose that $\mathscr{C}$ admits a generator matrix $\mathbf{G}$ of the form $\left(\mathbf{I}_k \mid \mathbf{A} \right)$ with $\mathbf{A} \in \mathbb{F}_q^{k\times(n-k)}$. $\mathbf{G}$ is said to be in* standard form*. Then, $\mathbf{H} = \left(-\mathbf{A}^T \mid \mathbf{I}_{n-k} \right)$ is a parity-check matrix of $\mathscr{C}$.*

Remark 1.3. *We will assume on many occasions that a linear code admits a generator matrix in standard form. It may be found by performing Gaussian elimination over any matrix whose rows generate the code. Such codes are said* systematic *in their k first positions. This will not be restrictive for the statements that we have to prove, as all codes of dimension k are* equivalent *to a systematic code (see below Definition 1.12 for code equivalence). By this, we mean that the encoding function $\mathbf{m} \in \mathbb{F}_q^k \mapsto \mathbf{mG}$ can be twisted by an invertible matrix $\mathbf{U} \in \mathbb{F}_q^{k\times k}$ so that the code spanned by $\mathbf{UG}$ is systematic in its k first positions.*

As we already stated, the purpose of a linear code is to transmit messages through a channel that may alter messages. More formally, if the original message is $\mathbf{m} = (m_0, \ldots, m_{n-1}) \in \mathbb{F}_q^n$, we describe the *received* message as a vector $\mathbf{y} \in \mathbb{F}_q^n$ of the form

$$\mathbf{y} = \mathbf{m} + \mathbf{e},$$

where $\mathbf{e} \in \mathbb{F}_q^n$ is a vector representing the errors made over each information symbol transmitted. For instance, $e_i = 0$ means that the i-th bit was transmitted correctly. This brings the following definition, that aims at quantifying the number of errors introduced by the channel.

Definition 1.4 (Hamming distance, Hamming weight)**.** *Let $\mathbf{v}, \mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$. The* Hamming weight *$w_H(\mathbf{v})$ of $\mathbf{v}$ is the number of indices i with $v_i \neq 0$:*

$$w_H(\mathbf{v}) = \#\left\{i \in \{0, \ldots, n-1\} \mid v_i \neq 0\right\}.$$

The Hamming distance *$d_H(\mathbf{x}, \mathbf{y})$ between $\mathbf{x}$ and $\mathbf{y}$ is the Hamming weight of their difference, that is the number of indices i with $x_i \neq y_i$,*

$$d_H(\mathbf{x}, \mathbf{y}) = w_H(\mathbf{x} - \mathbf{y}) = \#\left\{i \in \{0, \ldots, n-1\} \mid x_i \neq y_i\right\}.$$

In terms of errors, the Hamming distance between two vectors represents the Hamming weight of the error transforming one vector into the other one.

When considering a real-world transmission channel and specific code $\mathscr{C}$, a crucial question is to know how hard it is to recover the original message from the noisy message $\mathbf{y}$ when knowing that $\mathbf{x}$ is a codeword of the subspace $\mathscr{C}$. A first question to ask is the following: how many elements of $\mathscr{C}$ are close (in Hamming distance) to $\mathbf{y}$? If, for a low-error rate channel, a sphere centered in $\mathbf{y}$ contains only one codeword $\mathbf{x}$, then the original message is $\mathbf{x}$ with very high probability. A tool to ensure that all the spheres of a given radius centered in codewords do not contain other codewords is the minimum distance.

Definition 1.5 (Minimum distance of a code)**.** *Let $\mathscr{C}$ be a linear code over $\mathbb{F}_q^n$. The* minimum distance *$d(\mathscr{C})$ of $\mathscr{C}$ (or simply the* distance *of $\mathscr{C}$) is the minimum Hamming distance between its codewords. It is also the minimum Hamming weight of its non-zero codewords.*

$$d(\mathscr{C}) = \min_{\substack{\mathbf{x},\mathbf{y}\in\mathscr{C}\\ \mathbf{x}\neq\mathbf{y}}} \{w_H(\mathbf{x} - \mathbf{y})\} = \min_{\mathbf{v}\neq\mathbf{0}} \{w_H(\mathbf{v})\}.$$

The minimum distance of a code is a very important parameter because of the following theorem.

Theorem 1.6 ([MS86][Ch. 1, p. 10])**.** *A code $\mathscr{C}$ with minimum distance d can correct $\lfloor \frac{d-1}{2} \rfloor$ errors. $\lfloor \frac{d-1}{2} \rfloor$ is called the* capacity correction *of $\mathscr{C}$.*

We now have the main parameters describing a code: length, dimension and minimum distance. In the following, a code of length $n \geqslant 0$, dimension k, distance d, and defined over $\mathbb{F}_q^n$ is be called a $[n,k,d]_q$-code (or simply a $[n,k,d]$-code). Finding the minimum distance of a code is a fundamental question in coding theory (for instance in [Var97b, Var97a, DMS03]). In particular, Vardy proved in [Var97a] that it is an NP-hard problem. In many cases we only have lower bounds, which will be enough for our purpose. We may note only $[n,k,\geqslant d]$ or $[n,k]$.

1.1.2 Operations on Linear Codes

Let $\mathscr{C}$ be a $[n,k,t]_q$-linear code. We give here the definitions of classical codes associated to $\mathscr{C}$: the dual code, the subfield subcode, the extended code and the trace code. All of them can be easily described once a description of $\mathscr{C}$ is known.

Definition 1.7 (Dual Code [MS86][Ch. 1 p. 28])**.** *The* dual code $\mathscr{C}^\perp$ *is the subspace of* $\mathbb{F}_q^n$ *which is orthogonal to* $\mathscr{C}$*, that is*

$$\mathscr{C}^\perp = \left\{\mathbf{x} \in \mathbb{F}_q^n \mid \forall \mathbf{y} \in \mathscr{C}, \mathbf{x}\mathbf{y}^T = x_0y_0 + \cdots + x_{n-1}y_{n-1} = 0\right\}.$$

If $\mathscr{C}$ *has generator matrix* $\mathbf{G}$ *and parity-check matrix* $\mathbf{H}$*, then* $\mathscr{C}^\perp$ *has generator matrix* $\mathbf{H}$ *and parity-check matrix* $\mathbf{G}$*. For this reason,* $\mathscr{C}^\perp$ *has dimension* $n-k$*.*

Definition 1.8 (Subfield Subcode [MS86][Ch. 7 p. 207])**.** *Let* $q = p^s$ *with* p *prime and* $s > 0$*, and* $\mathscr{C}$ *be a linear code over an extension* $\mathbb{F}_{q^m}$ *of* $\mathbb{F}_q$ *(with* $m > 0$*). The* subfield subcode *of* $\mathscr{C}$ *over* $\mathbb{F}_q$*, denoted* $\mathscr{C}_{|\mathbb{F}_q}$*, is the set of all the codewords of* $\mathscr{C}$ *whose all coordinates lie in* $\mathbb{F}_q$*.*

$$\mathscr{C}_{|\mathbb{F}_q} = \left\{\mathbf{m} \in \mathbb{F}_q^n \mid \mathbf{m} \in \mathscr{C}\right\} = \mathscr{C} \cap \mathbb{F}_q^n.$$

Suppose $\mathscr{C}$ *has parity-check matrix an* $r \times n$ *full-rank matrix* $\mathbf{H} = (h_{i,j})_{\substack{0\leqslant i\leqslant r-1\\ 0\leqslant j\leqslant n-1}}$ *with entries* $h_{i,j}$ *in* $\mathbb{F}_{q^m}$*. Write each* $h_{i,j}$ *as a column vector of* m *coordinates over* $\mathbb{F}_q$*:* $h_{i,j} = (h_{i,j,0}, \ldots, h_{i,j,m-1})^T$*. Then* $\mathscr{C}_{|\mathbb{F}_q}$ *admits as parity-check matrix the* $(mr) \times n$ *matrix* $\mathbf{H}'$ *with entries in* $\mathbb{F}_q$ *obtained by replacing each entry of* $\mathbf{H}$ *by a column-vector of its* m *coordinates over* $\mathbb{F}_q$*:*

$$\mathbf{H}' = \left(\begin{array}{cccc}
h_{0,0,0} & h_{0,1,0} & \cdots & h_{0,n-1,0} \\
\vdots & & & \\
h_{0,0,m-1} & & & \\
\hline
h_{1,0,0} & & & \\
\vdots & & h_{i,j,\ell} & \\
\vdots & & & \\
\hline
\vdots & & & \\
h_{r,0,m-1} & & & h_{r,n-1,m-1}
\end{array}\right)$$

Thus, $\mathscr{C}_{|\mathbb{F}_q}$ *has dimension greater or equal to* $n - rm$ *(assuming* $rm \leqslant n$*).*

Definition 1.9 (Extended Code [MS86][Ch. 1 p. 27])**.** *The extended code of* $\mathscr{C}$*, denoted by* $\widetilde{\mathscr{C}}$*, is a code of length* $n+1$ *obtained by adding to each codeword* $\mathbf{m} = (m_0, \ldots, m_{n-1})$ *the coordinate* $-\sum_{j=0}^{n-1} m_j$*. If* $\mathscr{C}$ *has generator matrix* $\mathbf{G}$ *and parity-check matrix* $\mathbf{H}$*,* $\widetilde{\mathscr{C}}$ *admits respectively as generator matrix* $\widetilde{\mathbf{G}}$ *and parity-check matrix and* $\widetilde{\mathbf{H}}$ *given by:*

$$\widetilde{\mathbf{G}} = \left(\begin{array}{c|c} \mathbf{G} & \begin{array}{c} \vdots \\ -\sum_{j=0}^{n-1} g_{i,j} \\ \vdots \end{array} \end{array}\right), \qquad \widetilde{\mathbf{H}} = \left(\begin{array}{cc} 1 \ \cdots \ 1 & 1 \\ \hline \mathbf{H} & \begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \end{array}\right)$$

If the distance d of $\mathscr{C}$ is odd, then $\widetilde{\mathscr{C}}$ is a $[n+1, k, d+1]_q$-code.

Definition 1.10 (Trace Code [MS86][Ch. 7 p. 208])**.** *Let $\mathscr{C}$ be an $[n, k]$-code defined over a finite field $\mathbb{F}_{q^m}$ with q a prime power and $m > 0$. The* trace over $\mathbb{F}_q$ *of an element x of $\mathbb{F}_{q^m}$ is defined by*

$$\mathrm{Tr}_q(x) = x + x^q + \cdots + x^{q^{m-1}} \in \mathbb{F}_q.$$

The trace code of $\mathscr{C}$ *is the code over $\mathbb{F}_q$ defined by*

$$\mathrm{Tr}_q(\mathscr{C}) = \left\{(\mathrm{Tr}_q(m_0), \ldots, \mathrm{Tr}_q(m_{n-1})) \in \mathbb{F}_q^n \mid \mathbf{m} = (m_0, \ldots, m_{n-1}) \in \mathscr{C}\right\}.$$

When the context is clear, we will use only Tr *instead of* Tr_q.

Finally, we give the following result due to Delsarte linking on the dual of a subfield subcode and the dual of the original code over $\mathbb{F}_{q^m}$.

Proposition 1.11 ([MS86, Del75])**.** *Let $\mathscr{C}$ be a linear code over $\mathbb{F}_{q^m}$. Then, it holds that*

$$\left(\mathscr{C}_{|\mathbb{F}_q}\right)^{\perp} = \mathrm{Tr}_q(\mathscr{C}^{\perp}).$$

1.1.3 Code Permutation and Automorphism Group

As for many mathematical structures, codes can be endowed with a natural action of the symmetric group in the following way.

Definition 1.12 (Permutation acting on a code [MS86][Ch. 8 §5], Permutation equivalance of codes)**.** *Let $\mathscr{C}$ be a code of length n and σ be a permutation of the code positions (i.e. a permutation on the indices $\{0, \ldots, n-1\}$). The permutation σ acts on a codeword as follows*

$$\mathbf{m}^{\sigma} = (m_{\sigma(i)})_{0 \leqslant i \leqslant n-1}.$$

It holds that $\sigma(\mathscr{C}) = \{\mathbf{m}^{\sigma} \mid \mathbf{m} \in \mathscr{C}\}$ is a code of same length, dimension, and minimal distance as $\mathscr{C}$. Also, we shall say that $\sigma(\mathscr{C})$ is permutation-equivalent *to $\mathscr{C}$.*

We introduce a fundamental object associated to a code: the permutation group. By considering all the linear inversible applications of $\mathbb{F}_q^n$ we could define a more general structure (the automorphism group), but we will only need permutations in the Part II of this thesis.

Definition 1.13 (Automorphism group of a code restricted to permutations [MS86][Ch. 8 §5])**.** *Let $\mathbf{m} \in \mathscr{C}$ and σ be a permutation of the code positions. We shall say that σ is an* automorphism *of $\mathscr{C}$ if and only if $\mathbf{m}^{\sigma} \in \mathscr{C}$, for all $\mathbf{m} \in \mathscr{C}$. The set of automorphisms (restricted to permutations on the code positions) of $\mathscr{C}$ is a group, denoted by* $\mathrm{Aut}(\mathscr{C})$

We give the following useful result on the automorphism group of a code.

Proposition 1.14 (Automorphism group of the dual code)**.** *Let $\mathscr{C}$ be a linear code. Then, it holds that*

$$Aut(\mathscr{C}) = Aut\left(\mathscr{C}^{\perp}\right).$$

Example 1.15. *Let $\mathscr{C}$ be the code of length 5 over $\mathbb{F}_2$ generated by the rows of the matrix:*

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

The permutation of the code positions $i \mapsto i \oplus_5 1$ (with the convention $i \oplus_5 1 = (i+1) \mod 5$) is clearly a permutation of $\mathscr{C}$. Note that, in this example, $\mathbf{M}$ is not a generator matrix of $\mathscr{C}$ as it is not full-rank. Indeed, Gaussian elimination of $\mathbf{M}$ yields the following generator matrix:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Interestingly, in the previous example, we can see from $\mathbf{M}$ that $\mathscr{C}$ is invariant under $i \mapsto i \oplus_5 1$, but it is not obvious from $\mathbf{G}$. More generally, given a generator matrix of a linear code, determining its automorphism group can be done by using a method proposed by Leon [Leo82] in a time exponential in the code dimension, or an adaptation of Leon's method using the support splitting algorithm by Sendrier [Sen00], in which case the complexity is polynomial in the code length and exponential in the dimension of the intersection of the code with its dual (see for example [SS99, SS01b]).

Conversely, coding theorists studied the construction of linear codes with *known permutation group*. The specific case of alternant codes (see Definition 1.17 below) will be the center of Chapter 5.

1.2 Generalized Reed-Solomon and Associated Codes

We expose the definitions of the codes that will be useful for this thesis. Those codes are related to the *Reed-Solomon* codes, whose codewords are the evaluation of a polynomial of a bounded degree in points of a finite field.

Definition 1.16 (Reed-Solomon codes [MS86][Ch. 10 §2])**.** *Let $\mathbf{x} \in \mathbb{F}_{q^m}^n$ with $x_i \neq x_j$ for $i \neq j$, and set an integer $k \geqslant 0$. The* Reed-Solomon code *of* support $\mathbf{x}$*, denoted by* $\mathrm{RS}_k(\mathbf{x})$ *is the code*

$$\mathrm{RS}_k(\mathbf{x}) = \{(P(x_0), \ldots, P(x_{n-1})) \,|P \in \mathbb{F}_{q^m}[z]_{<k}\}.$$

Here, we will not use Reed-Solomon codes but codes with the same kind of representation: Generalized Reed-Solomon codes, alternant codes, and Goppa codes. They all share a common framework to decode efficiently (a bounded number) of errors.

Let $\mathbb{F}_q$ be a finite field of $q = p^s$ elements (p prime, and $s > 0$). We introduce the following Vandermonde-like matrices:

$$\mathbf{V}_t(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} y_0 & \cdots & y_{n-1} \\ y_0 x_0 & \cdots & y_{n-1} x_{n-1} \\ \vdots & & \vdots \\ y_0 x_0^{t-1} & \cdots & y_{n-1} x_{n-1}^{t-1} \end{pmatrix}, \tag{1.1}$$

where $\big(\mathbf{x} = (x_0, \ldots, x_{n-1}), \mathbf{y} = (y_0, \ldots, y_{n-1})\big) \in \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n$.

With suitable $\mathbf{x}$ and $\mathbf{y}$, the rows of such Vandermonde-like matrix define Generalized Reed-Solomon (GRS) codes (Definition 1.17).

Definition 1.17 (Generalized Reed-Solomon codes [RS60, MS86][Ch. 10 §8])**.** *Let* $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in (\mathbb{F}_{q^m})^n$ *where all* x_i*'s are distinct,* $\mathbf{y} = (y_0, \ldots, y_{n-1}) \in \big(\mathbb{F}_{q^m}^*\big)^n$ *and* t *a non-negative integer. The* Generalized Reed-Solomon code *of dimension* t*, denoted by* $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$*, is defined as follows:*

$$\mathrm{GRS}_t(\mathbf{x}, \mathbf{y}) = \big\{(y_0 Q(x_0), \ldots, y_{n-1} Q(x_{n-1})) \mid Q \in \mathbb{F}_{q^m}[z], \deg(Q) \leqslant t-1\big\}.$$

We shall call $\mathbf{x}$ *the* support *of the code, and* $\mathbf{y}$ *the* multipliers*.*

We will need in the following to consider the dual of a GRS code. It turns out to be also a GRS code, with description given by the next proposition.

Proposition 1.18 (Dual of a Generalized Reed-Solomon code [MS86][Ch.10 §8])**.** *Let* $\mathbf{x}$*,* $\mathbf{y}$ *and* t *be as in Definition 1.17. The dual of the code* $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$ *is given by*

$$\big(\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})\big)^{\perp} = \mathrm{GRS}_{n-t}(\mathbf{x}, \mathbf{y}'), \text{ with } y'_i = \frac{1}{y_i} \frac{1}{\prod\limits_{\substack{0 \leqslant j < n \\ j \neq i}} (x_j - x_i)}.$$

Alternant and Goppa codes can be viewed as subfield subcodes of duals of GRS codes over $\mathbb{F}_q$.

Definition 1.19 (Alternant Codes)**.** *Let* $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in (\mathbb{F}_{q^m})^n$ *where all* x_i*'s are distinct and* $\mathbf{y} \in (\mathbb{F}_{q^m}^*)^n$*. As for* GRS *codes,* $\mathbf{x}$ *is the* support *and* $\mathbf{y}$ *the* multipliers*. The code* $\mathcal{A}_t(\mathbf{x}, \mathbf{y}) = \big\{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{V}_t(\mathbf{x}, \mathbf{y})\mathbf{c}^T = \mathbf{0}\big\}$ *is called an* alternant code *of order* t*. In other words,* $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ *is the subfield subcode over* $\mathbb{F}_q$ *of the dual of* $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$*. Let* $\mathbf{y}'$ *be defined as in Proposition 1.18. The codewords of* $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ *have the following form:*

$$\mathcal{A}_t(\mathbf{x}, \mathbf{y}) = \big\{(y'_i Q(x_i))_{0 \leqslant i < n} \mid Q \in \mathbb{F}_{q^m}[z]_{\leqslant n-t-1} \text{ and } \forall 0 \leqslant i \leqslant n-1, y'_i Q(x_i) \in \mathbb{F}_q\big\}.$$

The code $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ *has length* n*, dimension* $\geqslant n - mt$*, and minimal distances* $\geqslant t+1$*. Moreover, knowing* $\mathbf{x}$ *and* $\mathbf{y}$*, up to* $\lfloor t/2 \rfloor$ *errors can be decoded efficiently.*

There exists a large family of supports and multipliers describing the same GRS and alternant codes. We give here a first family of such supports and multipliers. This property is proved by a stronger Theorem from [Dür87] that we state in Chapter 5.

Proposition 1.20. *Let* $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in (\mathbb{F}_{q^m})^n$ *where all* x_i*'s are distinct,* $\mathbf{y} \in (\mathbb{F}_{q^m}^*)^n$*, and* $t \geqslant 0$*. Let* $a, b, c, d \in \mathbb{F}_{q^m}^*$ *with* $ad - bc \neq 0$ *such that* $cx_i + d \neq 0$ *for all* $0 \leqslant i \leqslant n-1$*, and* $\lambda \in \mathbb{F}_{q^m}^*$*. Let* $\mathbf{x}'$ *and* $\mathbf{y}'$ *be defined for all* $0 \leqslant i \leqslant n-1$ *by:*

$$\begin{array}{rcl} x'_i & = & \frac{ax_i + b}{cx_i + d}, \\ y'_i & = & \lambda y_i (cx_i + d)^{t-1}. \end{array}$$

Then, it holds that $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y}) = \mathrm{GRS}_t(\mathbf{x}', \mathbf{y}')$*.*
Finally, $\widetilde{\mathbf{y}}$ *defined for all* $0 \leqslant i \leqslant n-1$ *by* $\widetilde{y}_i = \lambda y_i (cx_i + d)^{n-t-1}$*, it holds that* $\mathcal{A}_t(\mathbf{x}, \mathbf{y}) = \mathcal{A}_t(\mathbf{x}', \widetilde{\mathbf{y}})$*.*

Thanks to Proposition 1.11, we have the following characterization of the dual of an alternant code.

Lemma 1.21. *The dual $\mathcal{A}_t(\mathbf{x},\mathbf{y})^\perp$ of the alternant code $\mathcal{A}_t(\mathbf{x},\mathbf{y})$ of degree t and extension m over $\mathbb{F}_q$ is given by:*

$$\mathcal{A}_t(\mathbf{x},\mathbf{y})^\perp = \mathrm{Tr}_q\left(\mathrm{GRS}_t(\mathbf{x},\mathbf{y})\right) = \left\{\left(\mathrm{Tr}_q(c_0),\ldots,\mathrm{Tr}_q(c_{n-1})\right) \mid (c_0,\ldots,c_{n-1}) \in \mathrm{GRS}_t(\mathbf{x},\mathbf{y})\right\}.$$

Goppa codes form a subclass of alternant codes of particular interest for code-based cryptography (as we will explain in Chapter 2).

Definition 1.22 (Goppa Codes [MS86][Ch.12 §3)**.** *] Let $\mathbf{x}$ be as in Definition 1.19, and $g(z) \in \mathbb{F}_{q^m}[z]$ be of degree t satisfying $g(x_i) \neq 0$ for all $i, 0 \leqslant i \leqslant n-1$. We define the Goppa code over $\mathbb{F}_q$ associated to $g(z)$ as the code*

$$\mathscr{G}(\mathbf{x}, g(z)) = \mathcal{A}_t(\mathbf{x},\mathbf{y}), \text{ with } \mathbf{y} = g(\mathbf{x})^{-1}.$$

The dimension k of $\mathscr{G}(\mathbf{x}, g(z))$ satisfies $k \geqslant n - tm$. The polynomial $g(z)$ is called the Goppa polynomial*, and m is the* extension degree*. Equivalently, $\mathscr{G}(\mathbf{x}, g(z))$ can be defined as:*

$$\mathscr{G}(\mathbf{x}, g(z)) = \left\{\mathbf{c} = (c_0,\ldots,c_{n-1}) \in \mathbb{F}_q^n \mid \sum_{i=0}^{n-1} \frac{c_i}{z - x_i} \equiv 0 \bmod g(z)\right\}.$$

Goppa codes and alternant codes naturally inherit an efficient decoding algorithm that corrects up to $t/2$ errors that we will expose in Section 1.3. The interest for Goppa codes comes from the fact that a Goppa code defined over $\mathbb{F}_2$ allows to decode efficiently t errors instead of $t/2$. This comes from the following well-known result.

Theorem 1.23 ([MS86][p. 341], [Pat75])**.** *Let $\Gamma(z) \in \mathbb{F}_{2^m}[z]$ be a polynomial of degree t without multiple roots. The binary Goppa code $\mathscr{G}(\mathbf{x},\Gamma)$ is equal to the alternant code $\mathscr{A}_{2t}(\mathbf{x},\mathbf{y}^2)$, with $\mathbf{y}^2 = (\Gamma(x_i)^{-2})_{0\leqslant i\leqslant n-1}$. As a consequence, there exists a polynomial time algorithm decoding all errors of Hamming weight at most t in $\mathscr{G}(\mathbf{x},\Gamma)$ as soon as $\mathbf{x}$ and $\Gamma(z)$ are known.*

We also provide a useful property on the extended code of a Goppa code (proved for instance in [Ber00a, Proposition 2])

Proposition 1.24. *Let $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$ be a $[n,k]_q$-Goppa code with $t = \deg(g)$. Let $\mathbf{x}$ and $\mathbf{y}$ be such that $\mathscr{C} = \mathcal{A}_t(\mathbf{x},\mathbf{y})$. The extended code $\widetilde{\mathscr{C}}$ of $\mathscr{C}$ has parity-check matrix*

$$\widetilde{\mathbf{V}}_{t+1}(\mathbf{x},\mathbf{y}) = \begin{pmatrix} y_0 & \cdots & y_{n-1} & 0 \\ y_0x_0 & \cdots & y_{n-1}x_{n-1} & 0 \\ \vdots & & \vdots & \vdots \\ y_0x_0^{t-1} & \cdots & y_{n-1}x_{n-1}^{t-1} & 0 \\ y_0x_0^{t} & \cdots & y_{n-1}x_{n-1}^{t} & \frac{1}{\mathrm{LC}(g)} \end{pmatrix}$$

where $\mathrm{LC}(g)$ denotes the leading coefficient of g.

1.3 Decoding Tools for Alternant codes

To motivate the use of alternant codes in cryptography, we give a high-level description of the classical methods to decode errors in polynomial time for alternant codes (Alternant decoder) and binary Goppa codes (Patterson decoder). More detailed or more optimized descriptions can be found in [MS86][Ch.12 §9],[Ber, SKHN75]. These methods have in common the resolution of a *key equation*, which is an equation whose unkowns are univariate polynomials, and implies an extended Euclidean algorithm.

1.3.1 Alternant Decoder

Let $\mathscr{C} = \mathcal{A}_t(\mathbf{x}, \mathbf{y})$ be an alternant code over $\mathbb{F}_q$ with $\mathbf{x}, \mathbf{y} \in \mathbb{F}_{q^m}^n$ (Definitions 1.19, 1.22). Let $\mathbf{G}$ be a $k \times n$ be a generator matrix, and $\mathbf{H}$ be an $(n-k) \times n$ parity-check matrix.

Proposition 1.25 (Decoding tools)**.** *Suppose one wants to decode* $\mathbf{c} = \mathbf{mG} + \mathbf{e}$*, where* $w_H(\mathbf{e}) = w \leqslant \lfloor t/2 \rfloor$*. We write* $\mathbf{e} = (0, \ldots, 0, e_{i_1}, 0, \ldots, 0, e_{i_w}, 0, \ldots, 0)$*, with* $e_{i_j} \neq 0$*. The alternant decoder processes the following quantities:*

1. *The value* $\mathbf{s} = \mathbf{Hc}^T$ *is called the* syndrome*. Once* $\mathbf{H}$ *is fixed,* $\mathbf{s}$ *depends only on the error* $\mathbf{e}$ *and not on* $\mathbf{m}$*.*
2. *The* polynomial syndrome*, defined as follows (equalities proved below):*

$$S_{Alt,\mathbf{e}}(z) = \sum_{\ell=1}^{t-1} \left(\sum_{i=0}^{n-1} c_i y_i x_i^\ell \right) z^\ell = \sum_{\ell=1}^{t-1} \left(\sum_{j=1}^{w} e_{i_j} y_{i_j} x_{i_j}^\ell \right) z^\ell = \sum_{j=1}^{w} \frac{e_{i_j} y_{i_j}}{1 - z x_{i_j}} \mod z^t. \tag{1.2}$$

Moreover, if $\mathbf{H}$ *is in systematic form,* $S_{Alt,\mathbf{e}}(z)$ *can be deduced from the syndrome* $\mathbf{s}$ *thanks to*

$$S_{Alt,\mathbf{e}}(z) = \sum_{\ell=1}^{t-1} \left(\sum_{i=0}^{n-k-1} s_i y_i x_i^\ell \right) z^\ell. \tag{1.3}$$

Proof. Let $\mathbf{V}_t(\mathbf{x}, \mathbf{y})$ be defined as in (1.1). Equation (1.2) comes from the equality:

$$\mathbf{V}_t(\mathbf{x}, \mathbf{y})\mathbf{c}^T = \underbrace{\mathbf{V}_t(\mathbf{x}, \mathbf{y})\mathbf{G}^T}_{=\mathbf{0}} \mathbf{m}^T + \mathbf{V}_t(\mathbf{x}, \mathbf{y})\mathbf{e}^T = \mathbf{V}_t(\mathbf{x}, \mathbf{y})\mathbf{e}^T.$$

By identifying the coefficients, we obtain $\sum_{i=0}^{n-1} c_i y_i x_i^\ell = \sum_{i=0}^{n-1} e_i y_i x_i^\ell$ for all $0 \leqslant \ell \leqslant t-1$. We prove Equation (1.3), showing that the polynomial syndrome $S_{Alt,\mathbf{e}}(z)$ can be directly deduced from the received syndrome $\mathbf{s}$ when the parity-check matrix $\mathbf{H}$ is in systematic form. We write $\mathbf{s} = \mathbf{H}\tilde{\mathbf{s}}^T$, with $\tilde{\mathbf{s}} = (\mathbf{s}; 0, \ldots, 0)$. We have $\mathbf{H}\tilde{\mathbf{s}}^T = \mathbf{He}^T$, so that $\tilde{\mathbf{s}} - \mathbf{e} \in \mathcal{A}_t(\mathbf{x}, \mathbf{y})$ and $\mathbf{V}_t(\mathbf{x}, \mathbf{y})(\tilde{\mathbf{s}}^T - \mathbf{e}^T) = \mathbf{0}$. Therefore, for all $0 \leqslant \ell \leqslant t-1$, it holds that $\sum_{i=0}^{n-1} (\tilde{s}_i - e_i) y_i x_i^\ell = 0$. We deduce that

$$\sum_{i=0}^{n-k-1} s_i y_i x_i^\ell = \sum_{i=0}^{n-1} e_i y_i x_i^\ell.$$

The last equality of (1.2) is a classical series expansion. □

The purpose of the decoding algorithms for alternant codes is to recover, from the syndrome polynomial, polynomials whose roots are related to the support elements x_{i_j} in the error positions. More precisely, we look for the following polynomials.

Definition 1.26 (Error-locator polynomial, error-evaluator polynomial)**.** *Let* $\mathbf{e} = (0, \ldots, 0, e_{i_1}, 0, \ldots, 0, e_{i_w}, 0,$ *with* $e_{i_j} \neq 0$*. We define* $\sigma_{\mathbf{e}}(z)$ *the* error locator polynomial*,* $\sigma_{inv,\mathbf{e}}(z) = z^{\deg(\sigma_{\mathbf{e}})} \sigma_{\mathbf{e}}(z^{-1})$ *its*

reciprocal polynomial, and $\omega_{\mathbf{e}}(z)$ *the* error evaluator polynomial *by:*

$$\begin{aligned}
\sigma_{\mathbf{e}}(z) &= \prod_{j=1}^{w}(z - x_{i_j}) \\
\sigma_{inv,\mathbf{e}}(z) &= \prod_{j=1}^{w}(1 - zx_{i_j}) \\
\omega_{\mathbf{e}}(z) &= \sum_{j=1}^{w}\prod_{\substack{s=1\\ s\neq j}}^{w}(z - x_{i_s}) \\
\omega_{inv,\mathbf{e}}(z) &= \sum_{j=1}^{w} e_{i_j} y_{i_j} \prod_{\substack{s=1\\ s\neq j}}^{w}(1 - zx_{i_s})
\end{aligned}$$

When there is no ambiguity on the error vector, we will denote the error locator polynomial by $\sigma(z)$*, its reciprocal polynomial* σ_{inv} *and the error evaluator polynomial* $\omega(z)$*.*

Once the polynomial syndrome is known, the purpose is to find the error locator polynomial and error evaluator polynomial. Indeed, the roots of the error locator polynomials indicate the positions where $e_{i_j} \neq 0$. The value e_{i_j} of the error is then given by

$$e_{i_j} = -\frac{x_{i_j}\omega_{inv,\mathbf{e}}(x_{i_j}^{-1})}{y_{i_j}\sigma'_{inv,\mathbf{e}}(x_{i_j}^{-1})}, \tag{1.4}$$

where $\sigma'_{\mathbf{e}}$ is the derivative of $\sigma_{\mathbf{e}}$. In practice, the most costly part of the decoding consists in finding $\sigma(z)$ and $\omega(z)$. They are given by the next theorem.

Theorem 1.27 (Key equation for alternant decoder)**.** *Let* $S_{Alt,\mathbf{e}}(z), \sigma_{inv}(z)$ *and* $\omega_{\mathbf{e}}(z)$ *be defined as in Definition 1.25. Then, the following relation, called the* key equation*, holds:*

$$\sigma_{inv}(z)S_{Alt,\mathbf{e}}(z) = \omega_{inv}(z) \mod z^t. \tag{1.5}$$

Given a polynomial syndrome $S_{Alt,\mathbf{e}}(z)$*, then* $\sigma_{inv}(z)$ *and* $\omega_{inv}(z)$ *are the unique solution to* (1.5) *with* $\sigma_{inv}(0) = 1$*,* $\deg(\sigma_{inv}) \leqslant \lfloor t/2 \rfloor$*,* $\deg(\omega_{inv}) \leqslant \lfloor t/2 \rfloor - 1$*, and* $\deg(\sigma_{inv})$ *as small as possible.*

To solve the key equation, one needs to find polynomials $f(z), \sigma_{inv}(z)$ and $\omega_{inv}(z)$ with $\deg(\sigma_{inv}) \leqslant \lfloor t/2 \rfloor$, $\deg(\omega_{inv}) \leqslant \lfloor t/2 \rfloor - 1$ and

$$z^t f(z) + \sigma_{inv}(z)S_{Alt,\mathbf{e}}(z) = \omega_{inv}(z).$$

This is done thanks to the Extended Euclidean algorithm. This algorithm is well-known for computing the greatest common divisor of two polynomials (or integers) $a(z)$ and $b(z)$ and recovering the Bezout coefficients, that is two polynomials $u(z), v(z)$ such that

$$a(z)u(z) + b(z)v(z) = \gcd(a(z), b(z))$$

More precisely, by Extended Euclidean Algorithm (or EEA in short), we refer to Algorithm 1, where we also give as input a degree d and the algorithm stops as soon as we find $u(z), v(z)$ such that $a(z)u(z) + b(z)v(z)$ has degree lower or equal to d.

The purpose of starting the sequence $(r(z))$ by $r_{-1}(z)$ is to align the sequence $(q(z))$ with the iterations of the **while** loop: $q_i(z)$ is computed in the i-th iteration.

Therefore, the Key equation (1.5) can be solved by performing $\mathrm{EEA}(z^t, S_{Alt,\mathbf{e}}(z)), \lfloor t/2 \rfloor - 1)$, whose output is $\sigma_{inv}(z)$, some $f(z)$, and $\omega_{\mathbf{e}}(z)$. Note that, for decoding, the polynomial $f(z)$ is of no interest and Line 10 is removed from Algorithm 1 to gain efficiency.

Algorithm 1 Extended Euclidean Algorithm (EEA)

Input: $a(z), b(z), \deg(a) \geqslant \deg(b), d$
Output: $u(z), v(z), r(z)$ with $a(z)u(z) + b(z)v(z) = r(z)$ and $\deg(r) \leqslant d$

$r_{-1}(z) \leftarrow a(z), r_0(z) \leftarrow b(z)$
$u_{-1}(z) \leftarrow 1, u_0(z) \leftarrow 0$
$v_{-1}(z) \leftarrow 0, v_0(z) \leftarrow 1$
$i \leftarrow 0$
while $\deg(r_i(z)) > d$ **do**
  $i \leftarrow i + 1$
  $q_i \leftarrow r_{i-2}(z)/r_{i-1}(z)$ (quotient of the Euclidean division of $r_{i-2}(z)$ by $r_{i-1}(z)$)
  $r_i \leftarrow r_{i-2}(z) - q_i(z)r_{i-1}(z)$ (rest of the Euclidean division of $r_{i-2}(z)$ by $r_{i-1}(z)$)
  $u_i \leftarrow u_{i-2}(z) - q_i(z)u_{i-1}(z)$
  $v_i \leftarrow v_{i-2}(z) - q_i(z)v_{i-1}(z)$
end while
$N \leftarrow i$
return $u(z)_N, v_N(z), r_N(z)$

Theorem 1.28 (Resolution of the key equation, [MS86][Ch. 12 §9])**.** *Set $r_{-1}(z) = z^t$ and $r_0(z) = S_{Alt,\mathbf{e}}(z)$ and proceed Algorithm 1 until reaching an $r_N(z)$ such that $\deg(r_{N-1}(z)) \geqslant \lfloor t/2 \rfloor$ and $\deg(r_N(z)) \leqslant \lfloor t/2 \rfloor - 1$. Then the error locator and evaluator polynomials satisfying Equation (1.5) are given by*

$$\begin{array}{rcll} \sigma_{inv,\mathbf{e}}(z) & = & & \delta u_N(z), \\ \omega_{inv,\mathbf{e}}(z) & = & (-1)^N & \delta r_N, \end{array}$$

where $\delta \in \mathbb{F}_{q^m}^*$ is a constant such that $\sigma_{inv,\mathbf{e}}(0) = 1$.

Remark 1.29. *The error locator polynomial is deduced from $\sigma_{inv}(z)$* when the weight of the error vector is known. *Indeed, when 0 is a possible error position, one cannot determine with certainty the degree of $\sigma_{\mathbf{e}}(z)$ from the degree of its reciprocal polynomial $\sigma_{inv}(z)$. The reason is that for any polynomial $p(z) \in \mathbb{F}_{q^m}[z]$, $p(z)$ and $zp(z)$ have same reciprocal polynomial. Therefore, after computing a polynomial $\sigma_{inv}(z)$ of degree d, there are two possibilities :*

1. *the index α such that $x_\alpha = 0$ is not an error position, $\sigma_{\mathbf{e}}$ is not divisible by z, then $\deg(\sigma_{\mathbf{e}}) = \deg(\sigma_{inv})$ and $\sigma_{\mathbf{e}}(z)$ is equal to $z^{\deg(\sigma_{inv})}\sigma_{inv}(z^{-1})$,*
2. *α is an error position, and $\sigma_{\mathbf{e}}(z) = z^{\deg(\sigma_{inv})+1}\sigma_{inv}(z^{-1})$.*

In a noisy channel context, the number of error has no reason to be controlled and the only way we know for solving this ambiguity is to forbid the support $\mathbf{x}$ to contain 0. We will see another solution adapted to the cryptographic context. In [LS12], the authors assume that the error weight is fixed to a known value w so that the decoder can perform $\sigma_{\mathbf{e}}(z) = z^w\sigma_{inv}(z^{-1})$ in any case. We will discuss the security issues raised by this assumption in Chapter 8.

The successive steps of the decoding are summed up in Algorithm 2.

For a code defined over $\mathbb{F}_2$, Step 5 is not necessary, since $e_\ell = 1$ as soon as $\ell \in \{i_1, \ldots, i_w\}$.

Algorithm 2 Decoding Phases (Alternant decoder)

INPUT: A vector $\mathbf{c} \in \mathbb{F}_q^n$ with $\mathbf{c} = \mathbf{mG} + \mathbf{e}$ and $w_H(\mathbf{e}) \leqslant \lfloor t/2 \rfloor$,
A full description of $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$: $(\mathbf{x}, \mathbf{y})$ with support $\mathbf{x} \in (\mathbb{F}_q)^n, \mathbf{y} \in (\mathbb{F}_q^*)^n$, and $t \geqslant 0$.
OUTPUT: The error vector $\mathbf{e}$.

POLYNOMIAL SYNDROME DETERMINATION:
1: Determine the syndrome polynomial: $S_{Alt,\mathbf{e}}(z) = \sum_{\ell=1}^{t-1} \left(\sum_{j=1}^{n-1} c_i y_i x_i^\ell \right) z^\ell$.
ERROR LOCATOR POLYNOMIAL DETERMINATION:
2: Find two polynomials $\sigma_{inv}(z)$ and $\omega_{\mathbf{e}}(z)$ such as $\sigma_{inv}(z) S_{Alt,\mathbf{e}}(z) = \omega_{\mathbf{e}}(z) \mod g(z)$ and $\deg(\sigma_{inv}) \leqslant t, \deg(\omega_{\mathbf{e}}) < t$ by

$$\sigma_{inv}(z), \omega_{inv}(z) = \mathrm{EEA}(z^t, S_{Alt,\mathbf{e}}(z), \lfloor t/2 \rfloor - 1)$$

3: Compute: $\sigma_{\mathbf{e}}(z) = z^w \sigma_{inv}(z^{-1})$ (swap the coefficients of $\sigma_{inv}(z)$).
ERROR VECTOR DETERMINATION:
4: Find the i_j's with $\sigma_{\mathbf{e}}(x_{i_j}) = 0$.
5: Deduce the error vector $\mathbf{e}$: for $\ell \in \{0, \ldots, n-1\}$, if $\ell \in \{i_1, \ldots, i_w\}$ e_ℓ is given by Eq. (1.4) , else $e_\ell = 0$.
6: Return $\mathbf{e}$.

1.3.2 Patterson Method for Decoding Binary Goppa Codes

Of notable importance for code-based cryptography, there exists a polynomial-time algorithm solving t errors (instead of $\lfloor t/2 \rfloor$ as previously) on a codeword of an alternant code $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ when the code is actually a binary Goppa code $\mathscr{G}(\mathbf{x}, g(z))$ with $g(z)$ a square-free polynomial of degree t. Thanks to Theorem 1.23, we know that a codeword of $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ is a codeword of $\mathcal{A}_{2t}(\mathbf{x}, \mathbf{y}^2)$, so that the Algorithm 2 with inputs $(\mathbf{x}, \mathbf{y}^2)$ and $2t$ can be used to decode t errors. However, a more efficient method was proposed by Patterson in [Pat75]. It manipulates a different polynomial syndrom of degree t (instead of $2t$ with the alternant decoder), so we expect the complexity to be better.

Definition 1.30. *Let $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$ be a Goppa code with $g(z) \in \mathbb{F}_{2^m}[z]$ being a square-free polynomial of degree t. Let $\mathbf{G}$ be a $k \times n$ generator matrix, and $\mathbf{H}$ be an $(n-k) \times n$ parity-check matrix in systematic form. For a received word $\mathbf{c} = \mathbf{mG} + \mathbf{e}$ and a syndrome $\mathbf{s} = \mathbf{Hc}^T = \mathbf{He}^T$, we define the* polynomial syndrome *as*

$$S_{Gop,\mathbf{e}}(z) = \sum_{i=0}^{n-k-1} \frac{s_i}{z - x_i} = \sum_{i=0}^{n-1} \frac{e_i}{z - x_i} \mod g(z).$$

Equality holds for the same reason as with the alternant polynomial syndrome.
The Key equation is also different:

Proposition 1.31 (Key equation for Patterson decoding)**.** *With $\sigma_{\mathbf{e}}(z)$ defined as in Definition 1.26, the key equation writes as follows:*

$$S_{Gop,\mathbf{e}}(z) \sigma_{\mathbf{e}}(z) = \omega_{\mathbf{e}}(z) \mod g(z). \tag{1.6}$$

To solve Equation (1.6), Patterson's method consists in decomposing $\sigma_{\mathbf{e}}(z)$ into even and odd parts, which are squares of polynomials of degree at most $\lfloor t/2 \rfloor$ (since the polynomials' coefficients belong to a binary field $\mathbb{F}_{2^m}$):

$$\sigma_{\mathbf{e}}(z) = \sigma_1(z)^2 + z\sigma_2(z)^2.$$

Then, we rewrite (1.6) :

$$\sigma_1^2(z) = \left(S_{Gop,\mathbf{e}}(z)^{-1} + z\right)\sigma_2^2(z) \mod g(z).$$

We deduce that $\sigma_1(z)$ and $\sigma_2(z)$ satisfy

$$\tau(z)\sigma_2(z) = \sigma_1(z) \mod g(z)$$

with $\tau(z) = \sqrt{S_{Gop,\mathbf{e}}(z)^{-1} + z}$ (such a square root always exists because we work in a binary field). Therefore, $\sigma_1(z), \sigma_2(z)$ are exactly the output of $\mathrm{EEA}(g(z), \tau(z), \lfloor t/2 \rfloor)$. A notable difference with the alternant decoder is that the computation of $\tau(z)$ requires an inversion modulo $g(z)$, *i.e.* to find a polynomial $f(z)$ such that

$$f(z)S_{Gop,\mathbf{e}}(z) = 1 \mod g(z)$$

so this requires to perform the Extended Euclidean Algorithm $\mathrm{EEA}(g(z), S_{Gop,\mathbf{e}}(z), 1)$.

Once $\sigma_{\mathbf{e}}(z)$ has been recovered, it is sufficient to determine the error positions (that is, the roots of $\sigma_{\mathbf{e}}(z)$), since the errors' value is necessarily 1 for a binary code.

Algorithm 3 Decoding Phases (Patterson decoder)

INPUT: $n-k$-bit syndrome $\mathbf{s} = \mathbf{H}\mathbf{e}^T$ with $w_H(\mathbf{e}) \leqslant \lfloor t \rfloor$, private key $(\mathbf{x}, g(z))$
OUTPUT: The error vector $\mathbf{e}$.

POLYNOMIAL SYNDROME DETERMINATION:
1: Determine the syndrome polynomial: $S_{Gop,\mathbf{e}}(z) = \sum_{i=0}^{n-1} \frac{c_i}{z-x_i} \mod g(z)$
COMPUTATION OF $\tau(z)$:
2: Find $f(z)$ such that $f(z)S_{Gop,\mathbf{e}}(z) = 1 \mod g(z)$ by

$$f(z) = \mathrm{EEA}(g(z), S_{Gop,\mathbf{e}}(z), 0)$$

3: Set $\tau(z) = \sqrt{f(z) + z}$.
ERROR LOCATOR POLYNOMIAL DETERMINATION:
4: Find two polynomials $\sigma_1(z)$ and $\sigma_2(z)$ such as $\tau(z)\sigma_2(z) = \sigma_1(z) \mod g(z)$ and $\deg(\sigma_1(z)) \leqslant \lfloor t/2 \rfloor, \deg(\sigma_2(z))\lfloor t/2 \rfloor$ by

$$\sigma_1(z), \sigma_2(z) = \mathrm{EEA}(g(z), \tau(z), \lfloor t/2 \rfloor)$$

5: Deduce $\sigma_{\mathbf{e}}(z) = \sigma_1(z)^2 + z\sigma_2(z)^2$
ERROR VECTOR DETERMINATION:
6: ELP evaluation: $\mathbf{e} = (\sigma_{\mathbf{e}}(\mathbf{x}_0), \ldots, \sigma_{e'}(\mathbf{x}_{n-1})) \oplus (1, \ldots, 1)$.

Theorem 1.32 (Resolution of the Goppa key equation,[SKHN75])**.** *Set $r_{-1}(z) = g(z)$ and $r_0(z) = S_{Gop,\mathbf{e}}(z)$ and proceed and Euclidean algorithm until reaching an $r_N(z)$ such that $\deg(r_{N-1}(z)) \geqslant \lfloor t/2 \rfloor$ and $\deg(r_N(z)) \leqslant \lfloor t/2 \rfloor - 1$. Then the error locator and evaluator polynomials satisfying Equation (1.6) are given by*

$$\begin{array}{rcll} \sigma_{\mathbf{e}}(z) & = & & \delta u_N(z), \\ \omega_{\mathbf{e}}(z) & = & (-1)^N & \delta r_N. \end{array}$$

where $\delta \in \mathbb{F}_{q^m}^*$ is a constant which makes $\sigma_{\mathbf{e}}$ monic.

First Comparison of the Alternant and Goppa Decoders To assess accurately the complexity of decoding, we would have to give a detailed implementation of each part of the alternant and Patterson decoders. To give a common framework for both methods, we can split them as follows:

— polynomial syndrom computation
— resolution of the key equation (Patterson method or Alternant decoder)
— root-finding of the error-locator polynomial
— (for non-binary codes), computation of the error values

There are several choices to make when implementing each step, and the purpose of this paragraph is not to describe them. For details, we refer to implementation works, such as [Bis10, LS12, BCS13]. We will discuss specifically possible implementations of the resolution of the key equation in Chapter 8. To give a first comparison between the alternant decoder and Patterson decoder, we compare the EEA steps of both methods. We report the following result from [SKHN75].

Proposition 1.33. *Let N be the number of field multiplication performed during the execution of* $\mathrm{EEA}(a(z), b(z), d)$ *where $(a(z), b(z), d)$ are set either as in Step 2 of Algorithm 2 or Step 4 of Algorithm 3. For an input error vector $\mathbf{e}$ of weight w, it holds that*

$$N \leqslant 4\deg(a)w - w^2/2 + 13w/2$$

so that N is approximately bounded by $4\deg(a)w_H(\mathbf{e})$.

Remark that this proposition on the cost of the EEA executions holds only in the context of the resolution of a key equation. Indeed, in this case, the degree of the output polynomial $\sigma(z)$ (or $\sigma_{inv}(z)$) is controlled by the weight of the error introduced and this greatly helps when counting the number of operations. We will see in Chapter 8 that it can even be determined in advance with a suitable implementation. On the contrary, the EEA execution for polynomial inversion in the Patterson decoding cannot be as precisely controlled. A quick analysis shows that, for $\deg(a) \leqslant t$, the number of field multiplication is bounded by $2t^2$, so that we can give the following approximate comparison:

	Key eq	Syndrome Inversion	Total
Alternant Decoder	$8t^2$		$8t^2$
Patterson Decoder	$4t^2$	$2t^2$	$6t^2$

Figure 1.1 – Number of field multiplications required by the decoding steps.

Chapter 2

Code-Based Cryptography

The purpose of error-correcting codes was originally not to hide information but to protect it from transmission errors, following the works of Shannon on the noisy channel. It was only in 1978 that Robert McEliece described a trapdoor one-way function, that is, a function difficult to invert without knowing the private trapdoor, mainly consisting in an encoding procedure [McE78]. He exploited the fact that, as we saw in Section 1.3, a Goppa code can be efficiently decoded provided that some elements used to build the code are known, whereas encoding only requires the knowledge of a generator matrix. This principle can be applied with any code having such hidden trapdoor.

In this Chapter, we introduce formally McEliece encryption scheme, and we expose the difficult problems used in code-based cryptography to guarantee security. Namely, those are the hardness of decoding a random linear code (presented in Section 2.2), and the hardness of understanding the hidden structure of the public code (Sections 2.3 and 2.4). For each, we give some details about their current estimated complexities of resolution, which are necessary to scale secure parameters for real-word cryptosystems.

2.1 McEliece Encryption Scheme

The McEliece cryptosystem was proposed by Robert McEliece in 1978 [McE78]. It was the first public-key cryptosystem whose security relies on the hardness of the syndrom decoding (Problem 2.1). The trapdoor consists in a secret function able to decode in polynomial time a bounded number of errors. For such a trapdoor to exist, the public error-correcting code needs to have a specific structure, as the examples given in Chapter 1.

An instance of the McEliece cryptosystem is mainly defined by the choice of the public code. We will denote it $\mathscr{C}$, its length n and its dimension k. We give here a high-level description where $\mathscr{C}$ is not specified. In [McE78], McEliece suggested to instantiate $\mathscr{C}$ with a binary Goppa code $\mathscr{G}(\mathbf{x}, g(z))$ (with $\mathbf{x} \in \mathbb{F}_{q^m}^n$ and $g \in \mathbb{F}_{q^m}[x]$ of degree t, see Definition 1.22). The trapdoor T_t can be evaluated as soon as $\mathbf{x}$ and $g(z)$ are known.

Algorithm 4 McEliece Cryptosystem (1978)

PARAMETERS : Field size q, code length n, dimension k, and decoding capacity t
Message space: $\mathbb{F}_q^k$. Encrypted message space: $\mathbb{F}_q^n$.

KEYGEN : Choose a $[n, k]_q$ code $\mathscr{C}$ for which there exists a function T_t correcting t errors in polynomial time. $\mathbf{G}$ is a generator matrix of $\mathscr{C}$.

PRIVATE KEY : T_t a t-decoder for $\mathscr{C}$, $\mathbf{S}$ a random full rank $k \times k$ matrix , $\mathbf{P}$ a random $n \times n$ permutation matrix.
PUBLIC KEY : $\mathbf{G}_{pub} = \mathbf{SGP}$, t the correction capacity of the decoder T_t.

ENCRYPT :
1: Input $\mathbf{m} \in \mathbb{F}_q^k$.
2: Generate random $\mathbf{e} \in \mathbb{F}_q^n$ with Hamming weight t.
3: Output $\mathbf{c} = \mathbf{m}\mathbf{G}_{pub} + \mathbf{e}$.

DECRYPT :
1: Input $\mathbf{c} \in \mathbb{F}_q^n$.
2: Compute $\bar{\mathbf{m}} = T_t(\mathbf{c}\mathbf{P}^{-1})$
3: If decoding succeeds, output $\mathbf{S}^{-1}\bar{\mathbf{m}}$, else output $\perp$.

Remark about the private matrices P and S. The matrices $\mathbf{P}$ and $\mathbf{S}$ were proposed by McEliece to hide the structure of the private code and hopefully make the code generated by $\mathbf{G}_{pub}$ look random. An attacker willing to decrypt then has to tackle the problem of decoding a linear code. However, those matrices do not hide more structure than, for example, outputting a row reduced generator matrix of $\mathscr{G}(\mathbf{x}, g)$. Indeed, only knowing such a matrix, the decoding problem is as hard as for a random linear code. Moreover, the code generated by $\mathbf{G}_{pub}$ is only a permuted code of $\mathscr{G}(\mathbf{x}, g)$ and thus has the same Goppa polynomial and a permuted support. Therefore, they are considered to bring no additive security and are often dropped (or implicitly introduced, *e.g.* by considering a public matrix in systematic form). As for the matrix $\mathbf{P}$, it can be represented by a permutation of the support of the Goppa code: if $\mathbf{P}e_i = e_{\sigma(i)}$ (with $(e_0, \ldots, e_{n-1})$ a basis of $\mathbb{F}_2^n$ and σ a permutation of $\{0, \ldots, n-1\}$) and $\mathbf{G}$ is a generator matrix of $\mathscr{G}(\mathbf{x}, g)$, then $\mathbf{SGP}$ is a generator matrix of $\mathscr{G}(\mathbf{x}^\sigma, g)$, with $\mathbf{x}^\sigma = (x_{\sigma(i)})_{0 \leqslant i \leqslant n-1}$.

2.2 Syndrom Decoding Problem and Information Set Decoding

The hard problem underlying most code-based cryptosystems is the Syndrom Decoding Problem.

Problem 2.1 (Syndrom Decoding Problem)**.** *Let* $\mathbf{H}$ *be a* $(n-k)\times n$ *matrix over* $\mathbb{F}_q$, $\mathbf{s} \in \mathbb{F}_q^{n-k}$, *and* w *be an integer. The problem is to find* $\mathbf{e} \in \mathbb{F}_q^n$ *of weight* $w_H(\mathbf{e}) \leqslant w$ *such that* $\mathbf{s} = \mathbf{H}\mathbf{e}^T$.

Problem 2.1 is directly related to the general decoding problem for the following reason: pick an $[n, k]$ linear code $\mathscr{C}$ with generator matrix $\mathbf{G}$ and parity-check matrix $\mathbf{H}$. For a vector $\mathbf{m} \in \mathbb{F}_q^k$ and an error $\mathbf{e} \in \mathbb{F}_q^n$ with weight $w_H(\mathbf{e}) \leqslant w$, decoding consists in recovering $\mathbf{m}$ from the encoded message with errors $\mathbf{c} = \mathbf{m}\mathbf{G} + \mathbf{e}$. Knowing $\mathbf{H}$ one can compute the *syndrom* $\mathbf{s} = \mathbf{H}\mathbf{c}^T$, which depends only on $\mathbf{e}$ (as in Definition 1.25).

$$\mathbf{s} = \mathbf{H}\mathbf{c}^T = \underbrace{\mathbf{H}\mathbf{G}^T}_{=0}\mathbf{m}^T + \mathbf{H}\mathbf{e}^T = \mathbf{H}\mathbf{e}^T.$$

If one can solve the Syndrome Decoding Problem instance $\mathbf{s} = \mathbf{H}\mathbf{e}^T$, then $\mathbf{m}$ is recovered by solving the linear system with unknown $\mathbf{u} \in \mathbb{F}_q^n$: $\mathbf{u}\mathbf{G} = \mathbf{c} + \mathbf{e}$. Therefore, concerning the encryption primitives, the hardness of the Syndrom Decoding Problem prevents against message-recovery attacks.

The Syndrom Decoding Problem was proved to be NP-complete for binary linear codes in [BMvT78] and for q-ary codes by Barg (see for instance in [PBH98][Vol.1 Ch. 7 §4]). This NP-completeness is true when no information is known about the code spanned by $\mathbf{H}$ except $\mathbf{H}$. As we will see, code-based cryptosystems are often based on a trapdoor introduced in the code so that this decoding becomes easy (*ie* polynomial time) when a secret information is known. In those cases, it is crucial that the public description of the code (for instance, a generator or parity-check matrix) reveals no information about the secret trapdoor so that the Syndrome Decoding Problem is as hard as for a random linear code.

Though NP-complete, solving the Syndrome Decoding Problem has been an extensively studied problem. The most efficient algorithms are based on a framework called the Information Set Decoding. The general idea is the following: knowing a syndrome $\mathbf{s} \in \mathbb{F}_q^{n-k}$ and an $(n-k) \times n$ matrix $\mathbf{H}$, pick $\mathbf{H}'$ and $(n-k) \times (n-k)$ submatrix of $\mathbf{H}$ and compute $\mathbf{e}'$ such that $\mathbf{s} = \mathbf{H}'\mathbf{e}'$. If $\mathbf{e}'$ has weight lower or equal to w then Problem 2.1 is solved (complete $\mathbf{e}'$ with zeros to find a solution for $\mathbf{H}$). The overall complexity is

$$\mathcal{O}\left((n-k)^3 \frac{\binom{n}{w}}{\binom{n}{n-k}}\right) = \mathcal{O}\left((n-k)^3 \left(\frac{n}{n-k}\right)^w\right)$$

Many refinements of this method exist, *e.g* [LB88, Leo88, Ste88, CC98, BLP08, BLP11a, MMT11, BJMM12]. They have, as expected, exponential time complexity. The latest optimization was proposed in [MO15] and decodes in time complexity $2^{0.0473n}$ in the worst case (which occurs for an $[n, k]_2$ code with $k/n = 0.45$). All the quoted algorithms work for binary fields. An extension to $\mathbb{F}_q$ for all q's was designed by C. Peters in [Pet10]. Software implementations of this extension are publicly available on Peters' webpage [1] and are often used in recent articles. An up-to-date survey of the most recent evolutions can be found in [Meu12]. Finally, we point out that, as explained in [Ber10], a practicable quantum computer would bring new optimizations in Information Set Decoding methods. Thanks to the Grover

1. `http://christianepeters.wordpress.com/publications/tools/`

algorithm, a quantum computer would halve the exponent in the complexity (so $2^{n/40}$ for the quoted codes). Therefore, the key sizes would have to be quadrupled to keep the same security level.

2.3 The Decoder Recovery Problem

We already described the specific linear codes used to define the cryptographic primitives on which we will focus: alternant and Goppa codes (Definitions 1.19 and 1.22 in Chapter 1). As we mentioned in the previous Chapter, those codes are very structured. As a consequence, an additional assumption has to be made, jointly with the hardness of the Syndrome Decoding Problem, to ensure that the trapdoor is correctly hidden to a public user. In the encryption setting, this corresponds to the *key-security*. We state it as follows:

Problem 2.2 (Decoder Recovery Problem)**.** *Let $\mathscr{C}$ be an $[n, k, \geqslant 2t+1]_q$ code endowed with a function T_d decoding up to d errors in polynomial time. The Decoder Recovery Problem consists in finding any decoder T'_d (also decoding up to d errors in polynomial time) knowing only a public generator matrix of $\mathscr{C}$.*

This problem comes in a different flavor for each code family. Since 1978, various families of codes with polynomial decoder have been proposed for cryptographic use. The global purpose was to reduce the size of the public key compared to the binary Goppa codes of McEliece. For many of those proposals, the Decoder Recovery Problem turned out not to be hard. The first example is the Generalized Reed-Solomon codes, which were proved to be insecure by Sidelnikov and Shestakov in [SS92]. Then, large subcodes of GRS codes were suggested by Berger and Loidreau [BL05] and also broken (by Wieschebrink in [Wie06b, Wie10]). Sidelnikov proposed to use Reed-Muller codes [Sid94], which were shown to be insecure by Minder et al in [MS07]. Gabidulin codes (introduced in [Gab85]) were broken by Overbeck in [Ove05, Ove08]. Recently, algebraic-geometry codes with genus $g \geqslant 1$ were also broken very recently [CMCP14].

This illustrates that the choice of the public key is crucial for the security of the trapdoor. The second part of this thesis (Chapters 6 and 7) will present more recent examples of code families proposed for cryptographic use, but which turned out to be weak.

We detail below the method used to recover the trapdoor of a Generalized Reed-Solomon code, as it will be of interest for the second part of this thesis.

2.3.1 The Sidelnikov-Shestakov Attack

The Sidelnikov-Shestakov attack [SS92] is a classical attack against McEliece-like scheme instantiated with GRS codes [Nie86]. What follows is inspired by [Wie10]. Let $\mathbf{G}_{\mathrm{GRS}}$ be the generator matrix of $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$ (Definition 1.17) in systematic form. The goal of the attack is to recover the vectors $(\mathbf{x}, \mathbf{y}) \in (\mathbb{F}_{q^m})^n \times (\mathbb{F}_{q^m}^*)^n$ from $\mathbf{G}_{\mathrm{GRS}}$, which is of the form:

$$\mathbf{G}_{\mathrm{GRS}} = \left(\begin{array}{ccc|ccc} 1 & \dots & 0 & y_t Q_0(x_t) & \dots & y_{n-1} Q_0(x_{n-1}) \\ 0 & \ddots & 0 & \vdots & & \vdots \\ 0 & \dots & 1 & y_t Q_{t-1}(x_t) & \dots & y_{n-1} Q_{t-1}(x_{n-1}) \end{array}\right) = (g_{i,j})_{\substack{0 \leqslant i \leqslant t-1 \\ 0 \leqslant j \leqslant n-1}} \tag{2.1}$$

Proposition 2.3. *Let $\mathbf{G}_{\mathrm{GRS}} = \left(\mathbf{I}_t \mid \mathbf{U} \right) \in \mathbb{F}_{q^m}^{t \times n}$ be as in (2.1). Then the polynomials $\{Q_i\}_{0 \leqslant i \leqslant t-1}$ are of the form:*

$$Q_i(z) = \mathrm{LC}(Q_i) \prod_{\substack{0 \leqslant j \leqslant t-1 \\ j \neq i}} (z - x_j), \text{ where } \mathrm{LC}(Q_i) \neq 0 \text{ denotes the leading coefficient of } Q_i.$$

Also, no coefficient in the $\mathbf{U}$ *block can be zero.*

Proof. By definition, each row of $\mathbf{G}_{\mathrm{GRS}}$ is a codeword of $\mathrm{GRS}_t(\mathbf{x},\mathbf{y})$. Thus, each Q_i is a polynomial of degree $t-1$. Observe that $t-1$ distinct roots of the Q_i are already known, since $Q_i(x_j)=0$ for $0\leqslant j\leqslant t-1, j\neq i$. No coefficient in the $\mathbf{U}$ block can be zero, as it would provide another root for a Q_i. □

We can safely evaluate the quotients for $t\leqslant j\leqslant n-1$ and $0\leqslant i,i'\leqslant t-1$ as follows:

$$\frac{g_{i,j}}{g_{i',j}}=\frac{\mathrm{LC}(Q_i)(x_j-x_{i'})}{\mathrm{LC}(Q_{i'})(x_j-x_i)}. \tag{2.2}$$

The idea in [Wie10] is then to write equations of type (2.2) and to solve them to find the support. We apply the same idea, but in a more direct way here, as we give explicit formula linking the entries of $\mathbf{G}$ and the support elements.

Proposition 2.4. *Let* $\mathscr{C}=\mathrm{GRS}_t(\mathbf{x},\mathbf{y})$ *with* $\mathbf{G}=(g_{i,j})_{\substack{0\leqslant i\leqslant t-1\\0\leqslant j\leqslant n-1}}$ *being a generator matrix in systematic form. Let* $L_1=\left\{\frac{g_{0,i}}{g_{1,i}}\right\}_{t\leqslant i\leqslant n-1}$ *and*

$$L_2=\left\{\left(\frac{g_{i,t+1}}{g_{1,t+1}}-\frac{g_{i,t}}{g_{1,t}}\right)\left(\frac{g_{i,t+1}}{g_{0,t+1}}-\frac{g_{i,t}}{g_{0,t}}\right)^{-1}\right\}_{2\leqslant i\leqslant t-1}.$$

Then, a correct support for $\mathscr{C}$ *is given for any* $\alpha\in\mathbb{F}_{q^m}^*\setminus(L_1\cup L_2)$ *by* $\tilde{x}_0=0$, $\tilde{x}_1=1$, *and*

$$\begin{aligned}\tilde{x}_j &= \frac{1}{1-\frac{1}{\alpha}\frac{g_{0,j}}{g_{1,j}}} \text{ for } t\leqslant j\leqslant n-1,\\ \tilde{x}_j &= \frac{1}{1-\frac{1}{\alpha}\left(\frac{g_{j,t+1}}{g_{j,t+1}}-\frac{g_{j,t}}{g_{1,t}}\right)\left(\frac{g_{j,t+1}}{g_{0,t+1}}-\frac{g_{j,t}}{g_{0,t}}\right)^{-1}} \text{ for } 2\leqslant j\leqslant t-1.\end{aligned}$$

Proof. First, we show by a change of support that we can fix arbitrarily several values. Indeed, as for alternant codes in Proposition 1.20, for all a,b,c,d with $ad-bc\neq 0$ such that $cx_i+d\neq 0$ for all $0\leqslant i\leqslant n-1$, we rewrite $g_{i,j}=y_iP_j(x_i)$ as

$$y_iP_j(x_i)=y_i(cx_i+d)^{t-1}Q_j\left(\frac{ax_i+b}{cx_i+d}\right)=y_iH_j(\tilde{x}_i),$$

with P_j,H_j polynomials of degree $\leqslant t-1$. We want to fix arbitrarily two support elements, say $x_{i_0}=u_0, x_{i_1}=u_1$, and one quotient $\frac{\mathrm{LC}(H_{i_2}}{\mathrm{LC}(H_{i_3})}=k$, for $u_0\neq u_1$ in $\mathbb{F}_{q^m}$ and $k\in\mathbb{F}_{q^m}^*$. When $c\neq 0$, we observe that $\mathrm{LC}(H_j)=cQ_j(a/c)$, so that $\frac{\mathrm{LC}(H_{i_2})}{\mathrm{LC}(H_{i_3})}=\frac{\mathrm{LC}(Q_{i_2}}{\mathrm{LC}(Q_{i_3})}\frac{a/c-x_{i_3}}{a/c-x_{i_2}}$. That is, we impose on a,b,c,d the conditions:

$$\begin{aligned}ax_{i_0}+b &= u_0(cx_{i_0}+d),\\ ax_{i_1}+b &= u_1(cx_{i_1}+d),\\ \mathrm{LC}(Q_{i_2})(a-cx_{i_3}) &= k\mathrm{LC}(Q_{i_3})(a-cx_{i_2}).\end{aligned}$$

This linear system yields solutions with $c\neq 0$ when $\frac{\mathrm{LC}(Q_{i_2})}{\mathrm{LC}(Q_{i_3})}\neq k$. If $c=0$, $\frac{\mathrm{LC}(H_{i_2})}{\mathrm{LC}(H_{i_3})}=\frac{\mathrm{LC}(Q_{i_2})}{\mathrm{LC}(Q_{i_3})}$ so it already has the desired value. In practice, the desired a,b,c,d must be such that for all $0\leqslant i\leqslant n-1, cx_i+d\neq 0$. This is why, as we will see, some values k are not possible. Those will appear naturally as necessary conditions.

Thanks to this change of support, we fix in the following $x_0 = 0, x_1 = 1$ and note $\alpha = \frac{\mathrm{LC}(Q_0)}{\mathrm{LC}(Q_1)}$. In (2.2), we set $i = 0, i' = 1$, and obtain for $t \leqslant j \leqslant n-1$:

$$x_j = \frac{1}{1 - \frac{1}{\alpha}\frac{g_{0,j}}{g_{1,j}}}. \tag{2.3}$$

Then, we set $i' = 0$, pick $2 \leqslant i \leqslant t-1$ and $j = t$ to obtain:

$$\frac{g_{i,t}}{g_{0,t}}(x_t - x_i) = \frac{\mathrm{LC}(Q_i)}{\mathrm{LC}(Q_0)}x_t.$$

Now, let $j = t+1$:

$$\frac{g_{i,t+1}}{g_{0,t+1}}(x_{t+1} - x_i) = \frac{\mathrm{LC}(Q_i)}{\mathrm{LC}(Q_0)}x_{t+1}.$$

We obtain by quotienting (as the x_i's are all different and $x_0 = 0$),

$$x_i = \frac{x_{t+1}x_t\left(\frac{g_{i,t+1}}{g_{0,t+1}} - \frac{g_{i,t}}{g_{0,t}}\right)}{\frac{g_{i,t+1}}{g_{0,t+1}}x_t - \frac{g_{i,t}}{g_{0,t}}x_{t+1}}.$$

By using (2.3), we have for $2 \leqslant i \leqslant t-1$:

$$x_i = \frac{1}{1 - \frac{1}{\alpha}\left(\frac{g_{i,t+1}}{g_{1,t+1}} - \frac{g_{i,t}}{g_{1,t}}\right)\left(\frac{g_{i,t+1}}{g_{0,t+1}} - \frac{g_{i,t}}{g_{0,t}}\right)^{-1}}. \tag{2.4}$$

α is unkown, but to avoid zero denominators in (2.3) and (2.4), it is necessary that

$$\alpha \neq \frac{g_{0,i}}{g_{1,i}} \text{ for } t \leqslant i \leqslant n-1,$$

and for $2 \leqslant i \leqslant t-1$,

$$\alpha \neq \left(\frac{g_{i,t+1}}{g_{1,t+1}} - \frac{g_{i,t}}{g_{1,t}}\right)\left(\frac{g_{i,t+1}}{g_{0,t+1}} - \frac{g_{i,t}}{g_{0,t}}\right)^{-1}.$$

Therefore, if we avoid the forbidden values for α, we can reconstruct the support thanks to explicit expressions of the generator matrix's entries. □

Once a correct support is found, corresponding multipliers are found by solving a linear system as in Fact 4.1. This method comes from the algebraic attack that we introduce in Chapter 4. The complexity of recovering the secret structure is thus dominated by the linear system solving phase, so it is $O(k^3)$.

2.3.2 The Goppa Decoder Recovery Problem

The case of Goppa codes is of particular interest for us. They are amongst the few classes of codes which are still resistant to all the known attacks. We recall that they belong to the more general class of alternant codes which are all still considered secure. In other words, it is commonly assumed that no polynomial algorithm can solve the two following problems.

Problem 2.5 (Goppa Code Recovery Problem)**.** *Let $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$ be a Goppa code as defined in Definition 1.22. The Goppa Recovery Problem consists in finding $\mathbf{x}'$ and $g_1(z) \in \mathbb{F}_{q^m}[z]$ such that $\mathscr{C} = \mathscr{G}(\mathbf{x}', g_1(z))$ with the only knowledge of a generator matrix of $\mathscr{C}$.*

Problem 2.6 (Alternant Code Recovery Problem)**.** *Let $\mathscr{C} = \mathcal{A}_t(\mathbf{x}, \mathbf{y})$ be an alternant code as defined in Definition 1.19. The Alternant Code Recovery Problem consists in finding $\mathbf{x}'$ and $\mathbf{y}'$ such that $\mathscr{C} = \mathcal{A}_t(\mathbf{x}', \mathbf{y}')$ with the only knowledge of a generator matrix of $\mathscr{C}$.*

To explain why Goppa codes are still considered secure, we detail the current state-of-the-art of the attacks against the Goppa and Alternant Decoder Recovery Problem. The first one hinges on an exhaustive search of a secret element combined with a polynomial algorithm to restore the second secret. The second one is an algebraic attack, which will be the starting point of the second part of this thesis.

Support Splitting Algorithm.
The Support Splitting Algorithm (SSA), introduced by Sendrier, allows to recover in polynomial time a permutation (see Definition 1.12) sending a code on an other code of same length. More formally, the following theorem answers the problem of permutation-equivalence for codes over $\mathbb{F}_2$.

Theorem 2.7 (Support Splitting Algorithm, [Sen00])**.** *Let $\mathscr{C}$ and $\mathscr{C}'$ be two $[n, k]$ codes over $\mathbb{F}_2$. Then, there exists an algorithm which outputs a permutation $\sigma \in \mathcal{S}_n$ such that $\sigma(\mathscr{C}) = \mathscr{C}'$ (if it exists) in $O(n^3 + 2^h n\ell(n))$ with $h = \dim(\mathscr{C} \cap \mathscr{C}^\perp)$ and $\ell(n)$ being a parameter of the algorithm.*

In [Sen00], $\ell(n)$ is conjectured to be equal to $\ln(n)$. For Goppa codes (and for almost all linear codes) h is very small [Sen97]. This yields an algorithm with polynomial complexity in practice. Using the SSA algorithm, Sendrier and Loidreau designed an attack in [LS01] to recover the secret description of a binary Goppa code $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$. It assumes that the length of the Goppa code is the longest possible one for a given field $\mathbb{F}_{q^m}$, that is $n = q^m$. Such codes will be said to *full-support*. As the elements of the secret support $\mathbf{x}$ are all distinct, we know that $\mathbf{x}$ contains all the elements of $\mathbb{F}_{q^m}$. Therefore, if we set $\mathbf{x}^0 = (0, \alpha, \alpha^2, \ldots, \alpha^{q^m-1})$ (with α a generator of $\mathbb{F}_{q^m}$), then there exists $\sigma \in \mathcal{S}_n$ such that $(\mathbf{x}^0)^\sigma = \mathbf{x}$, and $\mathscr{G}(\mathbf{x}^0, g(z))$ is permutation-equivalent to $\mathscr{G}(\mathbf{x}, g(z))$. The attacks consists in trying all the possible Goppa polynomials. One needs to exhaust at most q^{mt} monic polynomials of degree t in $\mathbb{F}_{q^m}[z]$ (or in practice a bit less if the Goppa polynomial is supposed to be square-free or irreducible). We sum up the attack in Algorithm 5.

Algorithm 5 Loidreau-Sendrier Attack using SSA algorithm [LS01]

INPUT : A generator matrix $\mathbf{G}$ of a Goppa code $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$.
OUTPUT : A polynomial $g_1(z) \in \mathbb{F}_{q^m}[z]$ and a support $\mathbf{x}'$ such that $\mathscr{C} = \mathscr{G}(\mathbf{x}', g_1(z))$.

1: Set $\mathbf{x}^0 = (0, \alpha, \alpha^2, \ldots, \alpha^{q^m-1})$.
2: **repeat**
3: Pick $g_1 \in \mathbb{F}_{q^m}[z]$ of degree t \\Restrictions on $g_1(z)$ are possible [LS01]
4: Set $\mathscr{C}_1 = \mathscr{G}(\mathbf{x}^0, g_1(z))$
5: Apply the SSA algorithm with $\mathscr{C}_1$ and $\mathscr{C}$
6: **until** $\mathscr{C}_1$ and $\mathscr{C}$ are permutation-equivalent and $\sigma(\mathscr{C}_1) = \mathscr{C}$
7: Output $\mathbf{x}' = (\mathbf{x}^0)^\sigma$ and $g_1(z)$

Detailed analysis provided in [LS01] shows that only $2^{m(t-3)}/mt$ polynomials have to be tested. This brings down the complexity of Algorithm 5 to $n^3 2^{m(t-3)}/mt$ for a full-support code (that is, with $n = q^m$). When the code is not full support, then the possible support

sets also have to exhausted, introducing a $\binom{q^m}{n}$ factor in the complexity. This attack is thus essentially theoretical. Still, when designing cryptographic primitives using variants or subclasses of Goppa codes, it is important that the number of possible secret polynomials remain high. We will consider the case of non full-support codes in Chapter 4.

Remark that [Sen00] deals with codes over $\mathbb{F}_2$. Recent works ([SS13a, SS13b]) aimed at generalizing it to codes over $\mathbb{F}_q$ for any prime power $q \geqslant 2$. The problem becomes more complex because the notion of permutation-equivalence can be generalized in different ways. One can choose to keep the same action of $\sigma \in \mathcal{S}_n$ on a codeword $\mathbf{m} \in \mathbb{F}_q^n$, that is $\mathbf{m}^\sigma = (m_{\sigma(i)})_{0 \leqslant i \leqslant n-1}$. In this case, Theorem 2.7 extends without change. But this action of σ can be combined with a scaling of the coordinates. To do so, pick a vector $\mathbf{v} \in (\mathbb{F}_q^*)^n$, and define $\mathbf{m}^{(\mathbf{v},\sigma)} = (v_i m_{\sigma(i)})_{0 \leqslant i \leqslant n-1}$. In this case, the known algorithms to determine the existence of a couple $(\mathbf{v}, \sigma)$ linking two given codes $\mathscr{C}$ and $\mathscr{C}'$ have almost polynomial complexities for $q \in \{3, 4\}$ and exponential complexities for $q \geqslant 5$.

Regarding cryptographic cases, the schemes using codes defined over $\mathbb{F}_q$ with $q > 2$ all used the former action with only σ, so that the SSA algorithm applies.

An Algebraic Attack for the Alternant Recovery Problem. Faugère, Otmani, Perret and Tillich (FOPT) designed in [FOPT10a] the first algebraic modelling of the private support and multipliers of an alternant code, leading naturally to an algebraic attack. This work is the starting point of our algebraic attacks of Chapters 4, 6 and 7. In particular, we discuss it extensively in Chapter 4.

For now, we give in short the modelling: (FOPT) shows that the secret decoder of an alternant code can be recovered by solving the following system:

$$\left\{ g_{i,0} Y_0 X_0^\ell + \cdots + g_{i,n-1} Y_{n-1} X_{n-1}^\ell = 0 \ \mid \ i \in \{0, \ldots, k-1\}, \ell \in \{0, \ldots, t-1\} \right\} \tag{2.5}$$

where the unknowns $\mathbf{X} = (X_0, \ldots, X_{n-1})$, $\mathbf{Y} = (Y_0, \ldots, Y_{n-1})$ correspond to the secret support and multiplier respectively (see Definition 1.19). The $g_{i,j}$'s are the entries of the generator matrix.

2.4 The Goppa Distinguishing Problem

To build a complete security proof of cryptographic primitives using Goppa codes, designers rely not only on the hardness of the Syndrom Decoding Problem, but also on an other assumption, referred to as the *Goppa code indistinguishability*. This assumption was introduced in [Sen02][Ch. 9]. The global idea is to prove that if an attacker can efficiently decode a Goppa code without knowing the trapdoor and that Goppa codes behaves like random codes, then the attacker can efficiently decode a random code, thus violating the hardness of the Syndrom Decoding Problem.

Problem 2.8 (Goppa Distinguishing Problem)**.** *Let $\mathbf{H}$ be a $r \times n$ matrix with entries in $\mathbb{F}_q$. Decide whether there exists a Goppa code $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$ over $\mathbb{F}_q$ such as $\mathbf{H}$ is a parity-check matrix of $\mathscr{C}$.*

The Goppa indistinguishability assumption states that no polynomial-time algorithm solves the Goppa Distinguishing Problem.

[DMR11] shows that the Quantum Fourier Sampling, which is the quantum algorithm solving the integer factorization, cannot help solving the Goppa Distinguishing Problem for $[n, k]_q$ codes when $k/n \geqslant 1 - \frac{m}{\sqrt{5} q^{m/2}}$.

However, this does not rule out the existence of another quantum technique or classic algorithm. The Goppa indistinguishability assumption was proved to be wrong for codes with high dimensions in [FGO+13], where the following result is proved.

Theorem 2.9. *Let $n = q^m$. When q is fixed and m tends to infinity, there exists a polynomial-time algorithm solving the Goppa (and alternant) Distinguishing Problem for a $[n,k]_q$-code when*

$$\frac{k}{n} \geqslant 1 - \sqrt{\frac{2\,m\,\log_2 q}{q^m \log_2 m}}\Big(1 + o(1)\Big).$$

In particular, the security proof of the CFS signature scheme instantiated with Goppa codes that can be found in [Dal07] is rendered useless, as the codes used in this scheme all have high rates. On a more general point of view, this questions the trust we can have in the security proofs using Goppa indistinguishability.

Attacks using Code Distinguishers

In a more general context than the Goppa codes, distinguishing a specific family of codes from random codes is an important question in code based cryptograhy. The idea is to find a behaviour which is different for the studied family and for random codes. This different behaviour has to be testable efficiently. Once such distinguisher is found, it potentially leads to an attack (but not always, as the result previously mentionned). A striking example is that of the (already broken) GRS codes.

In [Wie10], Wieschebrink uses a very strong property of GRS codes to mount an attack on subcodes of GRS codes. This property focuses on the *square* code of a GRS code, defined by the coordinate-wise product of the codewords of a GRS code $\mathscr{C}$:

$$\mathscr{C} \star \mathscr{C} = \left\{ (m_0 m'_0, \ldots, m_{n-1} m'_{n-1}) \,\middle|\, \mathbf{m}, \mathbf{m}' \in \mathscr{C} \right\}.$$

Namely, if $\mathscr{C}$ is a subcode of $\mathrm{GRS}_k(\mathbf{x}, \mathbf{y})$ and $2k-1 \leqslant n-2$, then it is proved that $\mathscr{C} \star \mathscr{C}$ is a subcode of $\mathrm{GRS}_{2k-1}(\mathbf{x}, \mathbf{y} \star \mathbf{y})$. When equality holds the attacker has recovered a full GRS code and can apply the Sidelnikov-Shestakov attack. If $2k-1 > n-2$, the square code $\mathscr{C} \star \mathscr{C}$ is the full space $\mathbb{F}_q^n$ and this method fails. The trick is then to compute the square of a *shortened* code of $\mathscr{C}$, that is a subspace of $\mathscr{C}$ where some coordinates are required to be zero.

In any case, $\mathscr{C} \star \mathscr{C}$ has dimension $\leqslant 2k-1$. For a random code, the dimension of the square code is expected to be equal to the number of pairs of basis elements, that it $\frac{k(k+1)}{2}$ (or n if $n < \frac{k(k+1)}{2}$). Thus, this is a distinguishing property.

This distinguisher, along with the shortening method, was then used in a more powerful way and gave a lot of attacks. First, in [CGG+14], the authors break instances of GRS codes and their subcodes, and also the variants of [Wie06a, BL11, BBC+14]. Then, they extended it to algebraic geometry codes in[CMCP14]. Very recently ([COTGU15]), they broke with same ideas the latest proposition of GRS related codes [BBC+14].

Very recently, a polynomial-time algorithm using, among other, this distinguisher was proposed in a very specific case of the Goppa Recovery Problem in [COT14], namely Goppa codes of the form $\mathscr{G}(\mathbf{x}, g(z)^q)$ with $\mathbf{x} \in \mathbb{F}_{q^2}^n$ and $g(z) \in \mathbb{F}_{q^2}[z]$ (that will be referred to in Chapter 7 as wild Goppa codes over a quadratic extension).

Chapter 3

Gröbner Bases

The purpose of this chapter is to explain the tools that we used in this thesis to solve non-linear multivariate systems: Gröbner bases.

Gröbner bases are a very important object in commutative algebra, introduced by Bruno Buchberger in 1965 [Buc65]. Starting from a system of non-linear multivariate polynomials, Gröbner bases algorithms allow to re-write the system so as to obtain an equivalent system from which it is easy to deduce information about the solution set. They are a very active area of research in symbolic computation, with applications in many engineer sciences. Our purpose is not to explain the core of those algorithms. We rather give elements on the way they operate and their complexities in order to understand how to improve the algebraic modellings that we use for cryptanalytic purposes. The presentation that follows is mainly extracted from the Chapter 2 of [CLO07].

3.1 Basic Definitions

Let $\mathbb{K}$ be a field. In this thesis $\mathbb{K}$ will always be a finite field, but the results presented here are also true if $\mathbb{K}$ is a field of characteristic 0. We work in the ring $\mathbb{K}[X_1, \ldots, X_v]$ of polynomials in v variables. We recall that, for a finite set $f_1, \ldots, f_s$ of $\mathbb{K}[X_1, \ldots, X_v]$, the ideal generated by $f_1, \ldots, f_s$, denoted $\langle f_1, \ldots, f_s \rangle$, is the set

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i f_i, \mid h_1, \ldots, h_s \in \mathbb{K}[X_1, \ldots, X_v] \right\}.$$

Reciprocally, any ideal of $\mathbb{K}[X_1, \ldots, X_v]$ has a finite generating set. This is the Hilbert basis theorem.

Theorem 3.1 (Hilbert basis theorem)**.** *Let $\mathcal{I}$ be an ideal of $\mathbb{K}[X_1, \ldots, X_v]$. There exists a finite set of polynomials $f_1, \ldots, f_s \in \mathbb{K}[X_1, \ldots, X_v]$ such that $\mathcal{I} = \langle f_1, \ldots, f_s \rangle$.*

Definition 3.2 (Algebraic variety)**.** *Let $f_1, \ldots, f_s$ be polynomials in $\mathbb{K}[X_1, \ldots, X_v]$. We define $V(f_1, \ldots, f_s)$ the variety associated to $f_1, \ldots, f_s$ as the set of elemens $(a_1, \ldots, a_v)$ of $\bar{\mathbb{K}}^v$ such that*

$$\begin{aligned} f_1(a_1, \ldots, a_v) &= 0 \\ &\vdots \\ f_s(a_1, \ldots, a_v) &= 0 \end{aligned}$$

We will say that the elements of $V(f_1, \ldots, f_s)$ are the solutions of the algebraic system $\{f_1, \ldots, f_s\}$. We denote the set elements $(a_1, \ldots, a_v) \in \mathbb{K}^v$ which are solutions of $\{f_1, \ldots, f_s\}$ by $V_{\mathbb{K}}(f_1, \ldots, f_s)$. For $\mathcal{I} = \langle f_1, \ldots, f_s \rangle$ and $(a_1, \ldots, a_v) \in \mathbb{K}^v$, the following equivalence holds:

$$(a_1, \ldots, a_v) \in V_{\mathbb{K}}(f_1, \ldots, f_s) \iff \forall f \in \mathcal{I}, f(a_1, \ldots, a_v) = 0,$$

so that we also write $V(\mathcal{I})$ for $V(f_1, \ldots, f_s)$.

The central question for us will be, for given input polynomials $f_1, \ldots, f_s$, to solve the algebraic system $\{f_1, \ldots, f_s\}$, that is to give all its solutions. When the number of those solutions in $\bar{\mathbb{K}}$ is finite, an important value associated to an ideal of $V_{\mathbb{K}}(f_1, \ldots, f_s)$ is its *degree*, which counts the number of solutions with multiplicities. It is defined as follows.

Definition 3.3 (Degree of a 0-dimensional ideal)**.** *Let $\mathcal{I}$ be an ideal of $\mathbb{K}[X_1, \ldots, X_v]$ such that $V(\mathcal{I})$ is finite. Then, $\mathcal{I}$ is said to be* 0-dimensional*. The quotient $\mathbb{K}[X_1, \ldots, X_v]/\mathcal{I}$ is a vector space over $\mathbb{K}$ of finite dimension, and we set*

$$\deg(\mathcal{I}) = \dim_{\mathbb{K}} \left(\mathbb{K}[X_1, \ldots, X_v]/\mathcal{I} \right).$$

In this thesis, we will always look for solutions in a finite field $\mathbb{F}_q$ (with q a prime power). This can be expressed by including in the ideal the field equations $X_1^q - X_1, \ldots, X_v^q - X_v$. A consequence is that the associated ideals are always 0-dimensional.

Let's introduce some definitions.

Definition 3.4 (Monomial, term)**.** *A monomial is an element of $\mathbb{K}[X_1, \ldots, X_v]$ of the form*

$$X_1^{\alpha_1} \cdots X_v^{\alpha_v},$$

where $\alpha_1, \ldots, \alpha_v$ are non-negative integers. For shortness, we note $x^\alpha = X_1^{\alpha_1} \ldots X_v^{\alpha_v}$ with $\alpha = (\alpha_1, \ldots, \alpha_v)$. The degree of $X_1^{\alpha_1} \ldots X_v^{\alpha_v}$, denoted by $|\alpha|$, is equal to $|\alpha| = \alpha_1 + \cdots + \alpha_v$.

Any polynomial f of $\mathbb{K}[X_1, \ldots, X_v]$ can be written as finite linear combination (with coefficients a_α in $\mathbb{K}$) of monomials

$$f = \sum_{\alpha \in \mathbb{N}^v} a_\alpha x^\alpha,$$

a_α is called the coefficient of the monomial x^α is f. When $a_\alpha \neq 0$, then $a_\alpha x^\alpha$ is a term of f.

Definition 3.5 (Monomial ordering)**.** *A monomial ordering $>_{mon}$ on $\mathbb{K}[X_1, \ldots, X_v]$ is a relation on $\mathbb{N}^v$ satisfying:*

1. *$>_{mon}$ is a total ordering on $\mathbb{N}^v$, that is, for any $\alpha, \beta \in \mathbb{N}^v$, exactly one of the three statements $\alpha >_{mon} \beta$, $\alpha = \beta$, $\beta >_{mon} \alpha$ is true,*
2. *if $\alpha >_{mon} \beta$ and $\gamma \in \mathbb{N}^v$, then $\alpha + \gamma >_{mon} \beta + \gamma$,*
3. *$>_{mon}$ is well-ordering on $\mathbb{N}^v$ (that is, every non-empty subset of $\mathbb{N}^v$ has a smallest element under $>_{mon}$.*

For two monomials x^α and x^β of $\mathbb{K}[X_1, \ldots, X_v]$, we say that $x^\alpha >_{mon} x^\beta$ if and only if $\alpha >_{mon} \beta$.

Condition 2 guarantees the compatibility of $>_{mon}$ with multiplication by a monomial. For two monomials of $\mathbb{K}[X_1, \ldots, X_v]$ with $x^\alpha >_{mon} x^\beta$, then $x^\alpha x^\gamma >_{mon} x^\beta x^\gamma$ for any monomial x^γ.

Once a monomial ordering is chosen, we can define the leading monomial, coefficient and term a polynomial $f = \sum_\alpha a_\alpha x^\alpha$.

Definition 3.6 (Leading Monomial, Leading Coefficient, Leading Term)**.** *Let $f = \sum_\alpha a_\alpha x^\alpha$ be a polynomial of $\mathbb{K}[X_1, \ldots, X_v]$, and $>_{mon}$ a monomial ordering. Then,*

1. *the* **leading monomial** *of f, denoted $\mathrm{LM}_{>_{mon}}(f)$ is the maximal (for $>_{mon}$) monomial of f with non-zero coefficient,*
2. *the* **leading coefficient** *of f, denoted $\mathrm{LC}_{>_{mon}}(f)$ is the coefficient associated to the leading monomial of f,*
3. *the* **leading term** *of f, denoted $\mathrm{LT}_{>_{mon}}(f)$, is the term $\mathrm{LC}_{>_{mon}}(f)\mathrm{LM}_{>_{mon}}(f)$.*

We give the definitions of the monomials orderings the most useful for us.

Definition 3.7 (Lexicographical ordering, Graded reverse lexicographical ordering)**.** *Let $\mathbb{K}[X_1, \ldots, X_v]$ be a polynomial ring. We define:*

1. **Lexicographical order.** *For $\alpha, \beta \in \mathbb{N}^v$, we say that $\alpha >_{lex} \beta$ (and $x^\alpha >_{lex} x^\beta$) if and only if, in the vector difference $\alpha - \beta \in \mathbb{Z}^v$, the leftmost non-zero entry is positive.*
2. **Graded reverse lexicographical order.** *For $\alpha, \beta \in \mathbb{N}^v$, we say that $\alpha >_{grevlex} \beta$ (and $x^\alpha >_{grevlex} x^\beta$) if and only if $|\alpha| > |\beta|$ or $|\alpha| = |\beta|$ and the rightmost non-zero entry is negative.*

Example 3.8. *Let $f = 4xy^2x + 4z^2 - 5x^3 + 7x^2y^2 \in \mathbb{K}[x, y, z]$. If we write the terms of f in decreasing order, we obtain:*

— *with respect to the lexicographical order,*

$$f = -5x^3 + 7x^2z^2 + 4xy^2z + 4z^2,$$

and $\mathrm{LM}_{>_{lex}}(f) = x^3, \mathrm{LC}_{>_{lex}}(f) = -5, \mathrm{LT}_{>_{lex}}(f) = -5x^3$.

— *With respect to the graded reverse lexicographical order,*

$$f = 4xy^2z + 7x^2z^2 - 5x^3 + 4z^2,$$

and $\mathrm{LM}_{>_{grevlex}}(f) = xy^2, \mathrm{LC}_{>_{grevlex}}(f) = 4, \mathrm{LT}_{>_{grevlex}}(f) = 4xy^2$.

Thanks to the notion of monomial ordering, we can define the *Gröbner basis* associated to an ideal.

Definition 3.9 (Gröbner Bases)**.** *Let $\mathcal{I}$ be an ideal of $\mathbb{K}[X_1, \ldots, X_v]$ and $>_{mon}$ be a monomial ordering. A finite subset $G_{>_{mon}} = \{g_1, \ldots, g_t\}$ of $\mathcal{I}$ is said to be a Gröbner basis of $\mathcal{I}$ for the ordering $>_{mon}$ if*

$$\langle \mathrm{LT}_{>_{mon}}(g_1), \ldots, \mathrm{LT}_{>_{mon}}(g_t)\rangle = \langle \mathrm{LT}_{>_{mon}}(\mathcal{I})\rangle$$

where $\mathrm{LT}_{>_{mon}}(\mathcal{I})$ denotes the set of all the leading terms of elements of $\mathcal{I}$.

It is proved in [Buc65] that any ideal admits a Gröbner basis for any monomial ordering. To obtain unicity, extra conditions on the family G are necessary.

Theorem 3.10. *Let $\mathcal{I}$ be an ideal of $\mathbb{K}[X_1, \ldots, X_v]$ and $>_{mon}$ be a monomial ordering. Then,*

1. *Any Gröbner basis $G_{>_{mon}} = \{g_1, \ldots, g_t\}$ of $\mathcal{I}$ is a basis of $\mathcal{I}$, that is, any $f \in \mathcal{I}$ can be written as a sum*

$$f = \sum_{i=1}^{t} h_i g_i, \text{ with } h_1, \ldots, h_t \in \mathbb{K}[X_1, \ldots, X_v],$$

   in other words, it holds that $\mathcal{I} = \langle G_{>_{mon}}\rangle$.

2. *$\mathcal{I}$ admits a unique* **reduced Gröbner basis***, that is a finite subset $G_{>_{mon}} = \{g_1, \ldots, g_t\}$ of $\mathcal{I}$ such that*
   (a) $G_{>_{mon}}$ is a Gröbner basis of $\mathcal{I}$,
   (b) for all $p \in G_{>_{mon}}$, $\mathrm{LC}(p) = 1$,
   (c) for all $p \in G_{>_{mon}}$, no monomial of p belongs to $\langle \mathrm{LT}(G_{>_{mon}} \setminus \{p\})\rangle$.

Replacing the entries of an algebraic system $\{f_1, \ldots, f_s\}$ by a Gröbner basis of the ideal generated by those entries can be advantageous for the resolution, in particular with Gröbner bases for the lexicographical ordering.

Proposition 3.11 (Lex Gröbner Bases)**.** *Let $\mathcal{I}$ be a zero-dimensional ideal of $\mathbb{K}[X_1, \ldots, X_v]$. Then, a lexicographical Gröbner basis $G_{>_{lex}}$ of $\mathcal{I}$ has the following shape*

$$G_{>_{lex}} = \left\{ \begin{array}{r} g_{1,1}(X_1, X_2, \ldots, X_v) \\ \vdots \\ g_{1,\ell_1}(X_1, X_2, \ldots, X_v) \\ g_{2,1}(X_2, \ldots, X_v) \\ \vdots \\ g_{2,\ell_2}(X_2, \ldots, X_v) \\ \vdots \\ \vdots \\ g_{v-1,\ell_{v-1}}(X_{v-1}, X_v) \\ g_v(X_v) \end{array} \right.$$

That is, for $1 \leqslant i \leqslant v-1, 1 \leqslant j \leqslant \ell_i$, $g_{i,j}$ *is a polynomial in* $X_i, \ldots, X_v$, *and* g_v *is a univariate poolynomial in* X_v. *We say that the lexicographical order is useful for* **variable elimination** *purposes. For instance, in the last polynomial of the basis, the variables* $X_1, \ldots, X_{v-1}$ *do not appear. More generally, it holds that for any* $1 \leqslant u \leqslant v$, *a lexicographical Gröbner basis of the ideal* $I \cap \mathbb{K}[X_u, \ldots, X_v]$ *is the restricted bases* $\{g_{u,1}, g_{u,2}, \ldots, g_v\}$.

In this thesis, the base field is always a finite field, and solving the algebraic equation associated to the last polynomial g_{i_v} of a lexicographical Gröbner basis is exactly finding the roots of a univariate polynomial with coefficients in a finite field. This can be done efficiently (in complexity polynomial in v) and yields all the possible values of X_v for the elements of $V_{\mathbb{K}}(\mathcal{I})$. Once these values have been found, they can be substituted in the polynomials in (X_{v-1}, X_v), which becomes univariate polynomial equation, thus efficiently solved. Step-by-step, the solution of the system $\{g_1, \ldots, g_{i_n}\}$ (which are also the solutions of $\{f_1, \ldots, f_s\}$) can be found in time polynomial in v.

This illustrates the avantage of knowing the lexicographical Gröbner basis of an ideal generated by algebraic equations which is the most important for us. The crucial point now is to look at the algorithms computing the Gröbner basis of a given ideal.

3.2 General Strategy for solving polynomial system

The first algorithm computing Gröbner bases was proposed by Buchberger in [Buc65]. Although it is proved to terminate, nothing can be said a priori about the complexity.

In [Laz83], Lazard's theorem showed that the computation of a Gröbner basis of a system of equations can be done by performing linear algebra on a matrix representing the system. Before stating this theorem, we give an overview of the ideas behind it.

The first idea is to see Gröbner bases computation as a generalization of the resolution of linear systems by Gaussian elimination in the non-linear setting. Indeed, to solve a linear system

$$\begin{array}{rcl} b_{1,1}X_1 + \cdots + b_{1,v}X_v & = & 0 \\ & \vdots & \\ b_{m,1}X_1 + \cdots + b_{m,v}X_v & = & 0 \end{array},$$

we represent it matricially under the form

$$\mathbf{B}\begin{pmatrix} X_1 \\ \vdots \\ X_v \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}.$$

We perform a Gaussian elimination on $\mathbf{B}$ to obtain an equivalent system in triangular form

$$\begin{array}{rcl} b_{1,1}X_1 + \cdots + \quad b_{d+1}X_{d+1} + \ldots \quad + \ldots b_{1,v}X_v & = & 0 \\ \ddots & & \\ b_{d+1}X_{d+1} + \ldots \quad + \ldots b_{m,v}X_v & = & 0 \end{array}.$$

In other words, we replace a generating set of polynomials by another one where variables have been eliminated, so the triangular form of a linear system can be seen as the analogous of the lexicographical Gröbner basis of an ideal.

To transpose Gaussian elimination on non-linear system, we represent them with matrices. To describe all the polynomials belonging to an ideal $\mathcal{I}$ and with degree lower or equal to

a certain integer $d > 0$, we use the so-called Macaulay Matrix in degree d (Definition 3.12 below). All those polynomials are linear combinations of the rows of this matrix. Lazard's theorem states that there exists a degree D such that performing a special Gaussian elimination algorithm (*i.e.* with row and column swapping forbidden) on the Macaulay matrix in degree D yields a Gröbner basis of the ideal $\mathcal{I}$. Now let's give formal definitions.

Definition 3.12 (Macaulay Matrix)**.** *Let $f_1, \dots, f_s$ be polynomials of $\mathbb{K}[X_1, \dots, X_v]$ with degree respectively $d_1, \dots, d_s$, and $>_{mon}$ a monomial ordering. For d a positive integer, $M(d, v)$ is the set of all the monomials $m_1^{\leqslant d}, \dots, m_{\ell_d}^{\leqslant d}$ of $\mathbb{K}[X_1, \dots, X_v]$ of degree lower or equal to d and sorted in decreasing order. There are $\ell_d = \binom{v+d}{v}$ such monomials.*
The Macaulay matrix in degree d for $>_{mon}$, denoted $\mathrm{Mac}_{d,>_{mon}}(f_1, \dots, f_s)$ is a matrix with entries in $\mathbb{K}$ with $\ell_{d-d_1} + \cdots + \ell_{d-d_s}$ rows and ℓ_d columns. Its rows are indexed by the polynomials $m_k^{\leqslant (d-d_i)} f_i$ for all the input polynomials f_i and all the monomials $m_k^{\leqslant (d-d_i)}$ in degree $\leqslant d - d_i$, and its columns are indexed by all the monomials $m_j^{\leqslant d}$ in degree $\leqslant d$, sorted in decreasing order. The entry corresponding to $m_k^{\leqslant (d-d_i)} f_i$ and $m_j^{\leqslant d}$ is the coefficient of the monomial $m_j^{\leqslant d}$ in the polynomial $m_k^{\leqslant (d-d_i)} f_i$.

More precisely, if $f_i = \sum_\alpha a_\alpha x^\alpha$, $m_k^{(d-d_i)} = x^\beta$, then $\mathrm{Mac}_{d,>_{mon}}(f_1, \dots, f_s)$ contains a_α in the column corresponding to $m_j^{\leqslant d} = x^{\alpha+\beta}$.

$$
\begin{array}{cc}
 & >_{mon} \; m_j^{\leqslant d} \; >_{mon} \\
m_k^{\leqslant (d-d_i)} f_i & \begin{pmatrix} & \vdots & \\ \cdots & a_\alpha & \cdots \\ & \vdots & \end{pmatrix}
\end{array}
$$

Example 3.13. *Consider the polynomials of $\mathbb{K}[x, y]$ $f_1 = xy^2 - x^2 - 4y^2 + 4x$, $f_2 = x^2y + 2x^2 + 2y + 4$, $f_3 = -y^3 + xy - 4y^2 + 4x$. The matrix of $\langle f_1, f_2, f_3 \rangle$ in degree 3 for the grevlex ordering is*

$$
\begin{array}{c}
 \\ f_1 \\ f_2 \\ f_3
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc} x^3 & x^2y & xy^2 & y^3 & x^2 & xy & y^2 & x & y & 1 \end{array} \\
\begin{pmatrix}
0 & 0 & 1 & 0 & -1 & 0 & -4 & 4 & 0 & 0 \\
0 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 4 \\
0 & 0 & 0 & -1 & 0 & 1 & -4 & 4 & 0 & 0
\end{pmatrix}
\end{array}
$$

Linear combinations of the rows of $\mathrm{Mac}_{d,>_{mon}}(f_1, \dots, f_s)$ correspond to the same linear combinations on the corresponding polynomials. Thus, after performing a Gaussian elimination (without swapping the columns), the rows of the matrix obtained describe a new family of polynomials, which Lazard's theorem shows to have good properties (provided that the degree d is high enough).

Theorem 3.14 (Lazard's theorem [Laz83])**.** *Let $f_1, \dots, f_s$ be polynomials of $\mathbb{K}[X_1, \dots, X_v]$, there exists a degree D such that the rows of the row echelon form of $\mathrm{Mac}_{D,>_{mon}}(f_1, \dots, f_s)$ form a Gröbner basis of the ideal generated by $f_1, \dots, f_s$.*

This suggests an algorithm for computing a Gröbner basis of an ideal $\mathcal{I}$ generated by $\{f_1, \dots, f_s\}$: computing successively the row echelon form of the Macaulay matrices of $\{f_1, \dots, f_s\}$ successively in degree $\max(d_i), \max(d_i)+1, \dots$ and test at each step if we obtained a Gröbner

basis (this can be done efficiently). The computation ends as soon as we have reached the degree provided by Lazard's theorem, which is formally defined in Defintion 3.15.

Several algorithms using this idea were proposed. In this thesis, we use essentially the algorithm F4 (described in [Fau99]) implemented in the MAGMA software. A problem of this algorithm is that, when performing the Gaussian elimination on Macaulay matrices, a lot of linear combinations of the rows become zero rows and the computation was useless as it does not help finding a Gröbner basis. In [Fau02], a criterion is given to detect when those zero linear combination will appear and avoid to spend time computing them. Under some assumptions that are detailed in Section 3.2.1, it is proved that all the useless computations are avoided. An implementation of this algorithm can be found in the FGb software ([Fau10]), it was used to solve some systems in Chapter 6.

The complexity can be bounded by the cost of the Gaussian elimination on the biggest Macaulay matrix to consider, that is the Macaulay matrix in the maximal degree D reached during the computation. A crucial tool to bound this maximal degree is the so-called *degree of regularity.*

Definition 3.15. *Let $f_1, \ldots, f_s \in \mathbb{K}[X_1, \ldots, X_v]$ be homogeneous polynomials. We define the* degree of regularity *of $\mathcal{I}$, denoted $d_{reg}(\mathcal{I})$, as the smallest integer $d \geqslant 0$ such that the polynomials of degree d of $\mathcal{I}$ generate all the monomials of degree d in v variables. In other words,*

$$d_{reg}(\mathcal{I}) = \min \left\{ d \in \mathbb{N} \;\middle|\; \dim_{\mathbb{K}}(\{f \in \mathcal{I} \text{ with } \deg(f) = d\}) = \binom{v+d-1}{d} \right\},$$

where $\dim_{\mathbb{K}}$ is the dimension as a $\mathbb{K}$-vector space.
If $f_1, \ldots, f_s \in \mathbb{K}[X_1, \ldots, X_v]$ are not-homogeneous polynomials, then, for each $1 \leqslant i \leqslant s$, let $\hat{f}_i$ be the homogeneous part of highest degree of f_i. Then, we define $d_{reg}(\langle f_1, \ldots, f_s \rangle)$ as $d_{reg}(\langle \hat{f}_1, \ldots, \hat{f}_s \rangle)$

The degree of regularity bounds the maximal degree reached in the successive computation of the echelon form of the Macaulay matrices for a degree order. The cost of this echelon form uses the so-called *linear algebra* constant, which is known to satisfy $2 \leqslant \omega < 2.3727$ ([Wil12]). More precisely, when computing a Gröbner basis for the grevlex ordering, the following proposition holds.

Proposition 3.16 (Complexity bound for F_5 [BFS15])**.** *Let $\mathcal{I} = \langle f_1, \ldots, f_s \rangle \subset \mathbb{K}[X_1, \ldots, X_v]$ be a homogeneous zero-dimensional ideal. Let d_{reg} be its regularity degree. The complexity of computing a Gröbner basis for the grevlex ordering with the F_5 algorithm is in field operations asymptotically bounded when v grows to infinity by*

$$O\left(s d_{reg} \binom{v + d_{reg} - 1}{v}^{\omega} \right).$$

Thus, evaluating the degree of regularity is an important task to predict the complexity of Gröbner bases computation. We give more tools to do this in Paragraph 3.2.1. For now, we focus on solving polynomial systems.

In practice, grevlex Gröbner basis are often the easiest to compute, but are not the most convenient to find variety associated to an ideal. To do so, a lexicographical basis is required. To transform a Gröbner basis with to a given order into a basis for another order, so-called *changing order algorithms* are used. The most efficient one was introduced in [FGLM93] and is referred to as FGLM. It applies to zero-dimensional ideals.

Theorem 3.17 (FGLM algorithm [FGLM93, FGHR14]). *Let $\mathcal{I} = \langle f_1, \ldots, f_s \rangle$ be 0-dimensional ideal of $\mathbb{K}[X_1, \ldots, X_v]$ (Definition 3.3), and $G_{>_1}$ be a Gröbner basis of $\mathcal{I}$ for a monomial ordering $>_1$. Then, a Gröbner basis $G_{>_2}$ for an other monomial ordering can be computed in $O(v \deg(\mathcal{I})^\omega)$ operations in $\mathbb{K}$.*

We can now describe a complete a strategy of resolution of zero-dimensional polynomial systems.

Strategy of resolution

Thanks to Theorems 3.14 and 3.17 and Proposition 3.11, we explain in Algorithm 6 the strategy of resolution that we use in this thesis to solve algebraic systems over finite fields.

Algorithm 6 Strategy of resolution of a zero-dimensional ideal over a finite field

INPUT : An algebraic system $\{f_1, \ldots, f_s\}$ of $\mathbb{K}[X_1, \ldots, X_v]$ with $\mathbb{K}$ a finite field, $\mathbb{L}$ an extension of $\mathbb{K}$

OUTPUT : The algebraic variety $V_{\mathbb{L}}(f_1, \ldots, f_s)$.

1: Use F_4 or F_5 to obtain $G_{grevlex}$ a Gröbner basis of $\langle f_1, \ldots, f_s \rangle$ for the grevlex ordering (Definition 3.7)

2: Use the FGLM Algorithm with input $G_{grevlex}$ to compute

$$G_{lex} = \{g_{1,1}, \ldots, g_{1,\ell_1}, g_{2,1}, \ldots, g_{v-1,\ell_{v-1}}, g_v\}$$

a Gröbner basis of $\langle f_1, \ldots, f_s \rangle$ for the lexicographical ordering (as in Proposition 3.11).

3: Solve the algebraic equation $g_v(X_v) = 0$ over $\mathbb{L}$ to find the possible values for a_v and set

$$A_v = \{a_v \in \mathbb{L} \,|\, g_v(a_v) = 0\}.$$

4: **for** $i = v-1, \ldots, 1$ **do**

5: Set $A_i = \emptyset, j = 0$

6: **for** $(a_{i+1}, \ldots, a_v) \in A_{i+1}$. **do**

7: **repeat**

8: $j = j + 1$.

9: **until** $g_{i,j}(X, a_{i+1}, \ldots, a_v) \not\equiv 0$.

10: Solve the non-trivial univariate equation $g_{i,j}(X_i, a_{i+1}, \ldots, a_v)$ over $\mathbb{L}$ and set

$$A_{i+1} = A_{i+1} \cup \bigcup \left\{(a_i, \ldots, a_v) \in \mathbb{L}^{v-i+1} \,|\, g_{i,j}(X_i, a_{i+1}, \ldots, a_v) = 0\right\}.$$

11: **end for**

12: **end for**

13: Output A_1 the list of possible v-uples $(a_1, \ldots, a_v) \in \mathbb{L}^v$.

We apply Algorithm 6 on an example.

Example 3.18. *We use the same ring and polynomials as in Example 3.13. F_4 with inputs f_1, f_2, f_3 yields a Gröbner basis for the grevlex ording*

$$G_{grevlex} = \begin{cases} x^3 - 4x^2 + 2x - 8, \\ x^2y + 2x^2 + 2y + 4, \\ y^2 - x. \end{cases}$$

Then, a changing order algorithm gives a Gröbner basis for the lexicographical ordering

$$G_{lex} = \begin{cases} x - y^2, \\ y^5 + 2y^4 + 2y + 4. \end{cases}$$

So, to solve the system $f_1 = 0, f_2 = 0, f_3 = 0$, we solve the equation $y^5 + 2y^4 + 2y + 4 = 0$ in the desired field, and deduce x for each value of y.

We sum up the complexities of the steps of resolution in 3.1.

$$\boxed{\begin{array}{c}\text{Algebraic system}\\ \mathcal{I} = \langle f_1, \ldots, f_s \rangle\end{array}} \xrightarrow{O\left(sd_{reg}\binom{v+d_{reg}-1}{v}^{\omega}\right)} \boxed{\begin{array}{c}\text{Grevlex}\\ \text{Basis}\end{array}} \xrightarrow{O(v\deg(\mathcal{I})^{\omega})} \boxed{\text{Lex Basis}} \xrightarrow{O(v^3)} \boxed{V_{\mathbb{K}}(\mathcal{I})}$$

Figure 3.1 – Strategy of resolution of a zero-dimensional algebraic system

3.2.1 Complexity Asymptotics

In the previous strategy of reolution, the crucial element to predict the complexity is the degree of regularity. Understanding the degree of regularity requires to understand the structure of an ideal.

First, we need to determine if a system has a generic behaviour. To do this, Macaulay introduced in [Mac16] the notion of *regular* sequence, which applies to *square* systems of equations, *i.e.* systems with as many equations as unkowns. This notion was extended to *overdefined* systems (*i.e.* with more equations than unkowns) in [Bar04] by the notion of *semi-regularity*. Informally, it means that no useless computation are performed when computing a Gröbner basis with the F_5 algorithm.

Definition 3.19 (Semi-regular sequence [Bar04])**.** *Let $f_1, \ldots, f_s \in \mathbb{K}[x_1, \ldots, x_v]$ and $d_i = \deg(f_i)$. For $1 \leqslant i \leqslant s$, we define $\hat{f}_i$ the homogeneous part of highest degree of f_i. We say that $f_1, \ldots, f_s$ is a semi-regular if*

- *$\langle \hat{f}_1, \ldots, \hat{f}_s \rangle \neq \mathbb{K}[x_1, \ldots, x_v]$,*
- *for all $h \in \mathbb{K}[x_1, \ldots, x_v]$ and $1 \leqslant i \leqslant s$, if $h\hat{f}_i \in \langle \hat{f}_1, \ldots, \hat{f}_{i-1} \rangle$ and $\deg(h\hat{f}_i) \leqslant d_{reg}(\langle \hat{f}_1, \ldots, \hat{f}_s \rangle)$, then $h \in \langle \hat{f}_1, \ldots, \hat{f}_{i-1} \rangle$.*

For semi-regular sequences, the degree of regularity can be nicely characterized.

Proposition 3.20. *Let $f_1, \ldots, f_s \in \mathbb{K}[x_1, \ldots, x_v]$ a semi-regular sequence and $d_i = \deg(f_i)$ for $1 \leqslant i \leqslant s$. The degree regularity of $\langle f_1, \ldots, f_s \rangle$ is the index i of the first negative coefficient c_i of the series*

$$\frac{\prod_{i=1}^{s}(1 - z^{d_i})}{(1 - z)^v} = \sum_{i \geqslant 0} c_i z^i.$$

Thanks to this result, the degree of regularity can be computed explicitly for a given system, and it is possible to give asymptotic values for various categories of semi-regular systems. For example, for systems of $v + k$ polynomials in v variables, we have the following theorem.

Theorem 3.21 ([Bar04]). *The degree of regularity of a semi-regular sequence of $v+k$ polynomials of degree $d_1, \ldots, d_{v+k}$ in v variables behaves asymptotically when $v \longrightarrow \infty$ as*

$$d_{reg} = \sum_{i=1}^{v+k} \frac{d_i - 1}{2} - \alpha_k \sqrt{\sum_{i=1}^{v+k} \frac{d_i^2 - 1}{6}} + O(1)$$

where α_k is the biggest root of the k-th Hermite polynomial.

Now we give the example of systems of αv polynomials in v variables.

Theorem 3.22 ([Bar04]). *let $\alpha > 1$ be a constant. The degree of regularity of a semi-regular sequence of αv polynomials of degree d in v variables behaves asymptotically when $v \longrightarrow \infty$ as*

$$d_{reg} = \phi(z_0)v + O(v^{1/3})$$

where

$$\phi(z) = \frac{z}{1-z} - \frac{\alpha d z^d}{1 - z^d}$$

z_0 is the root of $\phi'(z)$ which minimizes $\phi(z_0) > 0$.

3.2.2 Degree of regularity in the algebraic modelling of alternant codes

The polynomial systems that we encounter in this thesis are not semi-regular in general, because they inherit from the object that they model a certain strucutre. An example of such structured system is provided by the algebraic modellings of alternant codes described in Chapter 4 (and used for cryptanalysis in Chapters 6 and 7).

Definition 3.23. *Let $f \in \mathbb{F}_q[X_1, \ldots, X_{v_X}, Y_1, \ldots, Y_{v_Y}]$. We shall say that f is* bi*-homogeneous of* bi-degree (d_1, d_2) *if:*

$$f(\alpha X_1, \ldots, \alpha X_{v_X}, \beta Y_1, \ldots, \beta Y_{v_Y}) = \alpha^{d_1}\beta^{d_2} f(X_1, \ldots, X_{v_X}, Y_1, \ldots, Y_{v_Y}), \ \forall (\alpha, \beta) \in \mathbb{F}_q \times \mathbb{F}_q.$$

Thus, f is bi-linear *if is is of* bi-degree $(1,1)$*. Also, if $p = \mathrm{char}(\mathbb{F}_q)$, then we shall say that f is* quasi bilinear *[FOPT10b] if it is of bi-degree (p^u, p^v) for $u, v \geqslant 0$*

To illustrate the improvements that such structure can bring to the resolution, we take the example of bilinear systems in $\mathbf{X} = (X_1, \ldots, X_{v_X})$ and $\mathbf{Y} = (Y_1, \ldots, Y_{v_Y})$. A bilinear system of $v_X + v_Y$ variables and with as many equations is in particular a quadratic system. For quadratic systems, we have the following bound.

Proposition 3.24 (Macaulay bound [Laz83]). *The degree of regularity of a square regular quadratic system in v variables is bounded by*

$$1 + v.$$

Thus, a square quadratic system has a degree of regularity bounded by $v_X + v_Y + 1$. However, recent advances in Gröbner bases computation (such as [FSS11]) show that better complexities can be obtained for bi-linear systems.

Proposition 3.25. *The degree of regularity of a square generic affine bi-linear system in $\mathbb{F}_q[X_1, \ldots, X_{v_X}, Y_1, \ldots, Y_{v_Y}]$ (as in Definition 3.23) is bounded by:*

$$2 + \min(v_X, v_Y).$$

This bound is sharp for a generic square affine bi-linear system and is much better than Macaulay's bound which bounds the degree of regularity by $1 + v_X + v_Y$.

3.3 Gröbner Bases and Modelling Strategy

In the rest of this manuscript, the strategy of Algorithm 6 will always be used. However, it is important to keep in mind some elements that influence the way we write the equations in our algebraic modellings:

1. In order to have an efficient changing-order step, it is highly preferable that the final system to solve be zero-dimensional. Generally, finding an infinite amount of solutions for a modelling to model a private key means that the key can be described more precisely.
2. The number of variables must be as low as possible, since all complexities rely on it.
3. As for semir-regular sequences, the more equations, the simpler the grevlex Gröbner basis computation is. Therefore, we try to write as many equations as possible.

The last two remarks raise a question: suppose that, to write some new equations satisfied by private elements, one needs to introduce additional variables, what will be the consequence on the solving complexity ? Today, no generic answer exists to this question. If the new equations reduce notably the degree of the associated ideal, then adding new equations can be of use. For the systems that we consider, we do not necessarily know the degrees of regularity, and in practice, it is important to make practical experiments to compare which approach is the most favorable one. Moreover, the systems are drawn from structured problems, and trying different modellings can reveal unexpected structure.

Part II

Contributions

Chapter 4

Algebraic attacks against McEliece-like schemes

The security of code-based cryptographic primitives relies, among other, on the difficulty for an attacker to recover the secret elements that allow to decode efficiently the errors, that is to solve the Decoder Recovery Problem (introduced in Section 2.3). We recalled in Paragraph 2.3.2 the two known attacks against alternant codes: the algebraic attack by Faugère, Otmani, Perret and Tillich ([FOPT10a], or FOPT in short) and, for the sub-family of Goppa codes, the SSA attack.

In this chapter, we first complete (in Section 4.1) the FOPT attack by introducing new equations describing more precisely the specificities of Goppa codes and binary Goppa codes (Table 4.1). In Section 4.2, we propose yet new systems which are more adapted to key-recovery for codes with high information rate (that is, with dimension very close to length). These new systems are crucial for the efficiency of the algebraic cryptanalysis of Chapter 6. Finally, in Section 4.4, we bound the complexity of resolution of the algebraic modellings to compare it with the complexity of the SSA attack. We show that the length of the code is the determinant factor in the choice of the most efficient attack: for *full-support* codes (with length $n \approx q^m$), the SSA attack is more efficient. On the contrary, to attack non full-support codes ($n \ll q^m$), algebraic methods are more efficient than the SSA attack.

4.1 Algebraic Descriptions of Alternant Codes and some subfamilies

Let's explain more in details how the algebraic system of FOPT was built. The public code is a q-ary alternant code $\mathscr{A}_t(\mathbf{x},\mathbf{y})$ defined by the public generator matrix $\mathbf{G} = (g_{i,j}) \in \mathbb{F}_q^{k\times n}$ (Definition 1.19). The couple of vectors $(\mathbf{x},\mathbf{y}) \in \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n$ permits to decode efficiently the public code (see Paragraph 1.3.1). Thus, $(\mathbf{x},\mathbf{y})$ is a key which is equivalent to the secret-key.

Let $\mathbf{X} = (X_0,\ldots,X_{n-1})$ and $\mathbf{Y} = (Y_0,\ldots,Y_{n-1})$ be two sets of variables corresponding to the unknown support $\mathbf{x}$ and multiplier $\mathbf{y}$ respectively. FOPT used the fact that $\mathbf{V}_t(\mathbf{X},\mathbf{Y})$ is a parity-check matrix (Definition 1.19) of the public-code. This means that $\mathbf{V}_t(\mathbf{X},\mathbf{Y})\mathbf{G}^T = (\mathbf{0})_{t\times k}$ holds and yields the following system:

$$\mathsf{A}_{\mathbf{X},\mathbf{Y}} = \bigcup_{\ell=0}^{t-1} \left\{ \sum_{j=0}^{n-1} g_{i,j} Y_j X_j^\ell \quad | \quad 0 \leqslant i \leqslant k-1 \right\}.$$

It is clear that $(\mathbf{x},\mathbf{y})$ is a solution of this system. Observe that $\mathsf{A}_{\mathbf{X},\mathbf{Y}}$ is very structured: the only monomials occurring are of the form $Y_j X_j^\ell$ with $0 \leqslant \ell \leqslant t-1$. Furthermore, $\mathsf{A}_{\mathbf{X},\mathbf{Y}}$ becomes linear as soon as the variables of the block $\mathbf{X}$ are fixed. We hence obtain kt linear equations with n unknowns $\mathbf{Y}$. The equations obtained by fixing the variables of $\mathbf{Y}$ in $\mathsf{A}_{\mathbf{X},\mathbf{Y}}$ are not directly linear. They are of the form $\sum_{j=0}^{n-1} g'_{i,j} X_j^\ell$ with $0 \leqslant i \leqslant k-1$. However, we can extract a linear system by only considering the equations with exponents $\ell = p^u$, with $0 \leqslant u \leqslant \lfloor \log_p(t-1) \rfloor$. We then apply the map $z \mapsto z^{q^m/p^u}$, which is additive in characteristic p, to this subset of equations. In other words, from the equation $\sum_{j=0}^{n-1} g'_{i,j} {X_j}^{p^u}$ we deduce that $\sum_{j=0}^{n-1} \left(g'_{i,j}\right)^{q^m/p^u} X_j^{q^m}$, which yields the linear equation $\sum_{j=0}^{n-1} \left(g'_{i,j}\right)^{q^m/p^u} X_j = 0$. This allows to obtain a linear system with $k\big(\lfloor \log_p(t-1) \rfloor + 1\big)$ equations.

Fact 4.1. *If the secret support* $\mathbf{x}$ *(resp. secret multiplier* $\mathbf{y}$*) is known then solving* $\mathsf{A}_{\mathbf{X},\mathbf{Y}}$ *reduces to solve a linear system of* kt *(resp.* $k(\lfloor \log_p(t-1) \rfloor + 1)$*) equations in* n *variables.*

As explained in [FOPT10b], the k linear equations involving only the set of variables $\mathbf{Y}$ can be used to eliminate some variables in the block $\mathbf{Y}$. By assumption, the public-code defined by $\mathbf{G}$ is of dimension k. Up to Gaussian elimination (and possibly reordering the positions), we can assume that $\mathbf{G}$ is in systematic form $\left(\mathbf{A} \mid \mathbf{I}_k \right)$ where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{A} = (a_{i,j}) \in \mathbb{F}_q^{k\times n-k}$. Thus, for all $\ell, 0 \leqslant \ell \leqslant t-1$:

$$Y_{n-k+i} X_{n-k+i}^\ell = -\sum_{j=0}^{n-k-1} a_{i,j} Y_j X_j^\ell, \quad \forall i, 0 \leqslant i \leqslant k-1. \tag{4.1}$$

By focusing on equations with $\ell = 0$, we construct a new polynomial system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ – where $\mathbf{Y}' = (Y_0,\ldots,Y_{n-k-1})$ – in which the variables $(Y_k,\ldots,Y_{n-1})$ have been eliminated:

$$\mathsf{A}_{\mathbf{X},\mathbf{Y}'} = \bigcup_{\ell=1}^{t-1} \left\{ \sum_{j=0}^{n-k-1} a_{i,j} Y_j \left(X_j^\ell - X_{n-k+i}^\ell \right) \mid 0 \leqslant i \leqslant k-1 \right\}.$$

4.1.1 Algebraic Description of Goppa Codes

The system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ only describes the fact the public key is the generator matrix of an alternant code. As mentionned in [BCMN10, BB11], when the public code is a Goppa code, the equivalent key $(\mathbf{x},\mathbf{y})$ vanishes $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ but also extra algebraic equations. Precisely, Definition 1.22 implies that there exists $\Gamma(z) = \sum_{\ell=0}^{t} \gamma_\ell z^\ell \in \mathbb{F}_{q^m}[z]$ of degree t such that $Y_j\Gamma(X_j) = \sum_{\ell=0}^{t} \gamma_\ell Y_j X_j^\ell = 1$. The coefficients $\gamma_0,\ldots,\gamma_t \in \mathbb{F}_q^m$ of $\Gamma(z)$ are unknown.

We know thanks to Proposition 1.24 of Chapter 1 that the extended code of $\mathcal{A}_t(\mathbf{x},\mathbf{y})$ with $\mathbf{y} = g(\mathbf{x})^{-1}$ has parity-check

$$\widetilde{\mathbf{V}_{t+1}}(\mathbf{x},\mathbf{y}) = \begin{pmatrix} y_0 & \cdots & y_{n-1} & 0 \\ y_0x_0 & \cdots & y_{n-1}x_{n-1} & 0 \\ \vdots & & \vdots & \vdots \\ y_0x_0^{t-1} & \cdots & y_{n-1}x_{n-1}^{t-1} & 0 \\ y_0x_0^{t} & \cdots & y_{n-1}x_{n-1}^{t} & \frac{1}{\gamma_t} \end{pmatrix}.$$

Let $\widetilde{\mathbf{G}}$ be the generator matrix of the extended code of $\mathbf{G}$, so $\widetilde{\mathbf{G}}$ is equal to $\mathbf{G} = \left(\begin{array}{c|c} \mathbf{A} & \mathbf{I}_k \end{array}\right)$ concatenated with the column

$$\begin{pmatrix} \vdots \\ -\left(1 + \sum_{j=0}^{n-k-1} a_{i,j}\right) \\ \vdots \end{pmatrix}.$$

We know that

$$\widetilde{\mathbf{V}_{t+1}}(\mathbf{x},\mathbf{y})\,\widetilde{\mathbf{G}}_{pub}^T = (\mathbf{0})_{(t+1)\times(n+1)}.$$

In this matrix-matrix product, the relations obtained from the first t rows of $\widetilde{\mathbf{V}_{t+1}}$ are the same as in $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$, but the $t+1-$th row yields for all $0 \leqslant i \leqslant k-1$ the relation:

$$\left(Y_{n-k+i}X_{n-k+i}^t + \sum_{j=0}^{n-k-1} a_{i,j}Y_jX_j^t\right) = \frac{1}{\gamma_t}\left(1 + \sum_{j=0}^{n-k-1} a_{i,j}\right). \tag{4.2}$$

So we introduce one variable Z for $\frac{1}{\gamma_t}$ and we obtain (after performing the same elimination as in $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$) a new polynomial system $\mathsf{G}_{\mathbf{X},\mathbf{Y}'}$ dedicated to q-ary Goppa codes: $\mathsf{G}_{\mathbf{X},\mathbf{Y}'} =$

$$\mathsf{A}_{\mathbf{X},\mathbf{Y}'} \cup \left\{ \sum_{j=0}^{n-k-1} a_{i,j}Y_j\left(X_j^t - X_{n-k+i}^t\right) = Z\left(1 + \sum_{j=0}^{n-k-1} a_{i,j}\right) \mid 0 \leqslant i \leqslant k-1 \right\}. \tag{4.3}$$

4.1.2 Algebraic Description of Binary Goppa Codes

The case of binary Goppa codes ($q = 2$) is even more specific. Such codes can be viewed as an alternant codes $\mathscr{A}_t(\mathbf{x},\mathbf{y})$ with $y_j = \Gamma(x_j)^{-1}$ for all $j, 0 \leqslant j \leqslant n-1$ but also described as a binary alternant codes $\mathscr{A}_{2t}(\mathbf{x},\mathbf{y}^2)$ (Theorem 1.23). This brings equations to the system $\mathsf{G}_{\mathbf{X},\mathbf{Y}'}$ which are defined by

$$Y_{n-k+i}^2X_{n-k+i}^\ell = -\sum_{j=0}^{n-k-1} a_{i,j}Y_j^2X_j^\ell$$

where $0 \leqslant i \leqslant k-1$ and $0 \leqslant \ell \leqslant 2t-1$. The equations obtained with $\ell = 2\ell'$ (with $0 \leqslant \ell' \leqslant t-1$) are the squares of equations of $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ (as in this case $a_{i,j}^2 = a_{i,j}$), so they bring no information. However, the equations in bi-degree $(2, 2\ell'+1)$ are new. This enables to define a specific algebraic system $\mathsf{McE}_{\mathbf{X},\mathbf{Y}'}$ dedicated to McEliece's cryptosystem:

$$\mathsf{McE}_{\mathbf{X},\mathbf{Y}'} = \mathsf{G}_{\mathbf{X},\mathbf{Y}'} \cup \bigcup_{\ell=0}^{t-1} \left\{ \sum_{j=0}^{n-k-1} a_{i,j} Y_j^2 \left(X_j^{2\ell+1} - X_{n-k+i}^{2\ell+1} \right) = 0 \mid 0 \leqslant i \leqslant k-1 \right\}. \tag{4.4}$$

4.2 Structural Elimination

We present a new method to eliminate some variables from the block $\mathbf{X}$ in our algebraic systems. The purpose is to avoid introducing too many variables. For instances, to model codes with long lengths n, dealing with systems containing $2n-k$ unkowns can be quite crippling for the resolution. In such situation, we explained in Paragraph 3.3 of Chapter 3 that reducing the number of variables can be helpful to ease the system solving phase.

Structural Elimination for the Alternant Modelling. The idea is to consider a suitable subset of the equations occurring in the system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$. More precisely the equations of bi-degree $(p^u, 1)$ (Definition 3.23 of Chapter 3) where $0 \leqslant u \leqslant \lfloor \log_p(t-1) \rfloor$. The only restriction is that the characteristic p has to be smaller than t.

This technique is inspired from ideas proposed to construct a distinguisher for alternant and Goppa codes [FGO+11, FGO+13]. The principle is to suitably combine monomials $Y_{n-k+i} X_{n-k+i}^{p^u}$, with $0 \leqslant u \leqslant \lfloor \log_p(t-1) \rfloor$ and $0 \leqslant i \leqslant k-1$. Thanks to (4.1), the trivial identities $\left(Y_{n-k+i} X_{n-k+i}^{p^u} \right)^p = Y_{n-k+i}^{p-1} Y_{n-k+i} X_{n-k+i}^{p^{u+1}}$ can be rewritten with $0 \leqslant u \leqslant \lfloor \log_p(t-1) \rfloor - 1$ and $0 \leqslant i \leqslant k-1$ as:

$$\left(- \sum_{j=0}^{n-k-1} a_{i,j} Y_j X_j^{p^u} \right)^p = \left(- \sum_{j=0}^{n-k-1} a_{i,j} Y_j \right)^{p-1} \left(- \sum_{j=0}^{n-k-1} a_{i,j} Y_j X_j^{p^{u+1}} \right). \tag{4.5}$$

Equation (4.5) only contains $X_0, \ldots, X_{n-k-1}$. We then set $\mathbf{X}' = (X_0, \ldots, X_{n-k-1})$. Thus, by cleaning-up relations (4.5), we define the system

$$\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'} = \bigcup_{u=0}^{\lfloor \log_p(t-1) \rfloor - 1} \bigcup_{i=0}^{k-1} \left\{ \sum_{j=0}^{n-k-1} X_j^{p^{u+1}} \left(a_{i,j}^p Y_j^p - a_{i,j} Y_j \left(- \sum_{j'=0}^{n-k-1} a_{i,j'} Y_{j'} \right)^{p-1} \right) \right\}.$$

We replaced equations of bi-degree $(p^u, 1)$ by ones of higher bi-degree (p^{u+1}, p). On the other hand, we reduced the number of variables from $2n-k$ to $2(n-k)$. In Section 6.2.3, we will see that for many practical parameters, and in particular parameters used in the signature schemes, the structural elimination allows to get significant practical improvements despite the degree increase.

We can also perform the elimination for Goppa codes (Equations (4.3)) and binary Goppa codes (Equations (4.4)).

Structural Elimination for Goppa Codes.

We can perform the elimination on Equations (4.3) when the Goppa polynomial has degree $t = p^\lambda$, which will be the case in Chapter 6 . To do so, develop

$$\left(Y_{n-k+i}X_{n-k+i}^{p^{(\lambda-1)}}\right)^p = Y_{n-k+i}^{p-1}\left(Y_{n-k+i}X_{n-k+i}^t\right)$$

thanks to (4.1) and (4.2). We obtain for $0 \leqslant i \leqslant k-1$,

$$\left(\left(1+\sum_{j=0}^{n-k-1} a_{i,j}\right)Z - \sum_{j=0}^{n-k-1} a_{i,j}Y_jX_j^{p^\lambda}\right)\left(\sum_{j=0}^{n-k-1} a_{i,j}Y_j\right)^{p-1} + (-1)^p\left(\sum_{j=0}^{n-k-1} a_{i,j}^pY_j^pX_j^{p^\lambda}\right) = 0 \tag{4.6}$$

So, we define the system $\mathsf{elimG}_{\mathbf{X}',\mathbf{Y}'}$ as the union of $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ and Equations (4.6). It describes Goppa codes and contains $2(n-k)+1$ variables.

Structural Elimination for Binary Goppa Codes.

In this paragraph, $q = 2$. To eliminate $X_{n-k}, \ldots, X_{n-1}$ in (4.4), we consider the equations in bi-degree $(2, 2u+1)$ for $0 \leqslant u \leqslant t-1$

$$Y_{n-k+i}^2X_{n-k+i}^{2u+1} = (Y_{n-k+i}X_{n-k+i})(Y_{n-k+i}X_{n-k+i}^{2u})$$

with Equation (4.4). We obtain for $0 \leqslant i \leqslant k-1$,

$$\left(\sum_{j=0}^{n-k-1} a_{i,j}Y_j^2X_j^{2u+1}\right) - \left(\sum_{j=0}^{n-k-1} a_{i,j}Y_jX_j\right)\left(\sum_{j=0}^{n-k-1} a_{i,j}Y_jX_j^{2u}\right) = 0 \tag{4.7}$$

So, we define the system $\mathsf{elimMcE}_{\mathbf{X}',\mathbf{Y}'}$ as the union of $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ and Equations (4.7). If $t = 2^\lambda$, Equations (4.6) can also be included. It describes binary Goppa codes and contains $2(n-k)+1$ variables.

4.3 Summary of the proposed Algebraic modellings

Table 4.1 summarizes the number of equations, number of variables in each block, and the structure of the different algebraic systems introduced in this part.

In practice, we exploit Proposition 1.20 to fix arbitrarily some values. We point out a difference with the exploitation of this property that we do in Sidelnikov-Shestakov attack (Paragraph 2.3.1). Indeed, in our Sidelnikov-Shestakov attack we fix four unkown values arbitrarily. Here, with our algebraic modelling, it is not possible to exploit all the degrees of freedom because using rational functions may lead to zero denominators which cannot be predicted. Therefore, we fix in $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ one of the Y_i''s and two X_i's to arbitrary values, which amounts to a simpler change of support $\mathbf{x}' = a\mathbf{x} + b$ and multipliers $\mathbf{y}' = c\mathbf{y}$ (for $a, c \in \mathbb{F}_{q^m}^*$ and $b \in \mathbb{F}_{q^m}$).

Also, some variables have to be added to be sure that the variety associated to $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ has few solutions. Indeed, the definition of a correct support implies that all the support elements are distinct and that all multipliers are non-zero, so we have to remove parasite solutions corresponding to $X_i = X_j$ and $Y_j = 0$. A classical way to do so that is to introduce new variables u_{ij} and v_i and add to $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ equations of the form $u_{ij} \cdot (X_i - X_j) + 1 = 0$ and $v_i \cdot Y_i + 1 = 0$. In practice, to avoid adding too many new variables, one can add only a few of them.

	$\#\mathbf{X}$	$\#\mathbf{Y}'$	Equations
$\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$	$n-2$	$n-k-1$	$k\cdot(t-1)$ eq. of bi-degree $(u,1)$, with $1\leqslant u\leqslant t-1$.
$\mathsf{G}_{\mathbf{X},\mathbf{Y}'}$	$n-2$	$n-k$	$\#\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ eq. $+\ k$ eq. of bi-degree $(t,1)$.
$\mathsf{McE}_{\mathbf{X},\mathbf{Y}'}$	$n-2$	$n-k$	$\#\mathsf{G}_{\mathbf{X},\mathbf{Y}'}$ eq . $+\ k\cdot t$ eq. of bi-degree $(2u+1,2)$ with $0\leqslant u\leqslant t-1$.
$\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$	$n-k-2$	$n-k-1$	$k\cdot\lfloor\log_p(t-1)\rfloor$ eq. of bi-degree (p^u,p), with $1\leqslant u\leqslant\lfloor\log_p(t-1)\rfloor$.
$\mathsf{elimG}_{\mathbf{X}',\mathbf{Y}'}$	$n-k-2$	$n-k$	$\#\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ eq. $+\ k$ eq. of bi-degree (t,p) with $t=p^\lambda$.
$\mathsf{elimMcE}_{\mathbf{X}',\mathbf{Y}'}$	$n-k-2$	$n-k$	k eq. of bi-degree (t,p) with $t=2^\lambda$ $+\ k\cdot\lfloor t\rfloor$ eq. of bi-degree $(2u+1,2)$, with $0\leqslant u\leqslant\lfloor t\rfloor$.

Table 4.1 – Algebraic key-recovery systems for McEliece-like cryptosystems. The notation # denotes the size of the set considered. For $\mathsf{G}_{\mathbf{X},\mathbf{Y}'}, \mathsf{McE}_{\mathbf{X},\mathbf{Y}'}, \mathsf{elimG}_{\mathbf{X}',\mathbf{Y}'}, \mathsf{elimMcE}_{\mathbf{X}',\mathbf{Y}'}$, the extra-variable Z is counted in the $\mathbf{Y}$ block. For $\mathsf{elimG}_{\mathbf{X}',\mathbf{Y}'}$ and $\mathsf{elimMcE}_{\mathbf{X}',\mathbf{Y}'}$, we assume that $t=p^\lambda$.

4.4 Complexity Bounds & Asymptotics

We provide complexity estimates of the algebraic attacks by using Fact 4.1: suppose all the multipliers have been guessed correctly (by exhaustive search), then the support is found by solving a linear system, and the secret key is known. Using Equations 4.1 with $\ell=0$, the number of multipliers to guess correctly is not n but $n-k=mt$. We obtain for this guess-and-solve strategy a complexity $\mathrm{GuessAlg}(n,t,m,q)$ given by

$$\mathrm{GuessAlg}(n,t,m,q)=(q^m)^{mt}\cdot\mathrm{poly}(n)=q^{m^2t}\cdot\mathrm{poly}(n).$$

This has to be compared with the complexity of the SSA-based attack (mentionned in Chapter 2): first one has to enumerate all the possible support sets ($\binom{q^m}{n}$ possibilities) and the irreducible Goppa polynomials (q^{mt}/t choices), then apply the SSA algorithm:

$$\mathrm{SSA}(n,t,m,q)=\binom{q^m}{n}(q^{mt}/t)\cdot\mathrm{poly}(n).$$

To compare asymptotically the complexities, we compute the ratio:

$$\mathrm{R}(n,t,m,q)=\frac{\mathrm{SSA}(n,t,m,q)}{\mathrm{GuessAlg}(n,t,m,q)}=\binom{q^m}{n}\frac{q^{mt-m^2t}}{t},$$

and its logarithm

$$\log(\mathrm{R}(n,t,m,q))=\log\binom{q^m}{n}+(mt-m^2t)\log(q)-\log(t).$$

In what follows, we evaluate its asymptotic behaviour in different scenarios. The idea is to determine whether $\log\binom{q^m}{n}$ grows faster than m^2t.

When making parameters grow, we have to respect some constraints. First, we must keep $n \leqslant q^m$, otherwise $\mathbb{F}_{q^m}$ does not contain enough elements to build a Goppa code of length n. We also need to have $n - mt \geqslant 0$ otherwise the code considered is the zero code (containing only 0).

4.4.1 Full-Support Codes

Full-support codes are typically those used to instantiate CFS signature schemes [FS11]. In this case, the support set contains all the elements of the finite field $\mathbb{F}_{q^m}$, so that $n = q^m$. For full-support codes, the enumeration factor is equal to 1. The SSA attack has a complexity exponential in mt, whereas the naive guess-and-solve strategy has complexity exponential in m^2t. So, it is clear that the SSA attack is better.

To have an idea of the best attack for *almost* full-support codes, we propose to link m and n by $n = q^m - c$, with c a constant. In this case we obtain $\binom{q^m}{n} = \binom{q^m}{c} \sim q^{mc}/c!$ and

$$\mathrm{R}(n,t,m,q) \underset{n\to\infty}{\sim} \frac{q^{m(c+t-mt)}}{c!t} \underset{\substack{n,m\to\infty \\ m\sim\log_q(n)}}{\longrightarrow} 0.$$

The SSA attack is asymptotically better than the guess-and-solve strategy when $n = q^m - c$ and m grows.

4.4.2 Non Full-Support codes

Non full-support codes are of particular interest for us, as all the codes that we consider in Chapters 6 and 7 fall into this category. In the following we look at three different situations where the code is *not* full-support. First, we fix n and m and make q tend to infinity. Then, we fix q and make n grow. When n grows to infinity, we have to assume that m also grows since we must keep $n \leqslant q^m$ (otherwise $\mathbb{F}_{q^m}$ does not contain enough elements to build a Goppa code of length n). We propose then two families of parameters: $n = \alpha q^m$ with α and q constant, which corresponds to codes with high rates (tending to 1), and $n = cmt$ with c and t constant, describing codes with constant rates.

4.4.2.1 Growing Base Field Size.

We suppose that n, m and t are fixed. The code dimension k is also constant. We consider codes over $\mathbb{F}_q$ where q grows to infinity. This hypothesis has the benefit of raising moderately the public key size, equal to $mt(n-mt)\log_2(q)$. This has to be compared with the key size of a full-support code ($mt(q^m - mt)\log_2(q)$), which grows polynomially when q tends to infinity.

Also, growing q makes ISD harder. This led Peters to propose Goppa codes over $\mathbb{F}_{31}$ with 2^{128} security and smaller keys than binary Goppa codes [Pet10].

When q grows to infinity, we estimate the binomial $\binom{q^m}{n}$ to $q^{mn}/n!$ and we obtain

$$\mathrm{R}(n,t,m,q) \underset{q\to\infty}{\sim} \frac{q^{m(n+t-mt)}}{n!t} = \frac{q^{m(k+t)}}{n!t} \underset{q\to\infty}{\longrightarrow} \infty.$$

The guess-and-solve strategy is asymptotically better than the SSA attack when q grows.

4.4.2.2 Codes with High Rates.

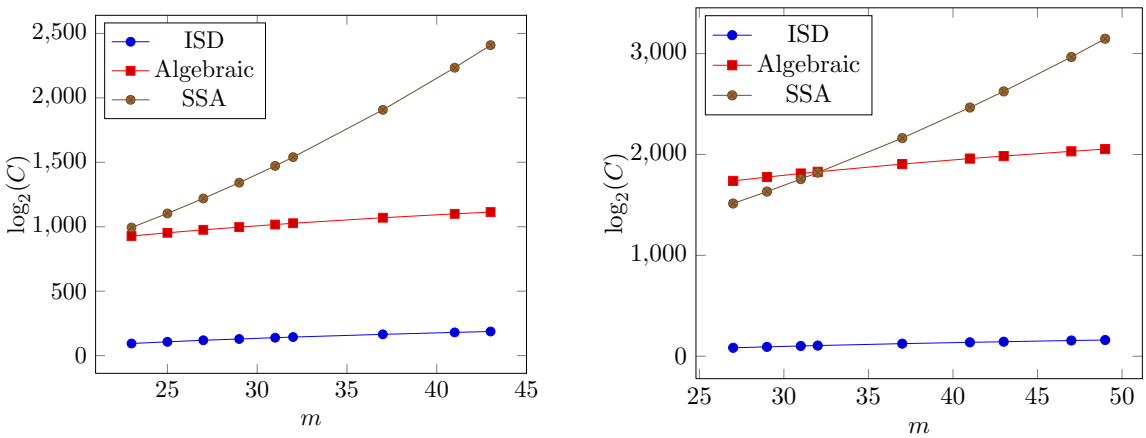
If m and n are related by $n = \alpha q^m$, with α and q constant, and m grows to infinity, the codes considered have high information rates $\frac{k}{n} = \frac{\alpha q^m - mt}{\alpha q^m}$. In this case, we evaluate the logarithm of the ratio of the complexities:

$$\log(\mathrm{R}(n,t,m,q)) = \log \binom{q^m}{\alpha q^m} + (mt - m^2 t)\log(q) - \log(t).$$

We perform an asymptotic development of $\log \binom{q^m}{\alpha q^m}$ by using $\log(u!) = u\log(u) - u + o(u)$. We obtain $\log(\binom{q^m}{\alpha q^m}) = H(\alpha)q^m + o(q^m)$, with $H(\alpha) = -(\alpha\log(\alpha) + (1-\alpha)\log(1-\alpha))$ being a positive function for $0 < \alpha < 1$, so that

$$\log(\mathrm{R}(n,t,m,q)) \underset{\substack{n,m\to\infty \\ n=\alpha q^m}}{\sim} H(\alpha)q^m \longrightarrow +\infty.$$

The guess-and-solve attack is asymptotically better than the SSA attack. We illustrate this in Figure 4.1.



$\alpha = 0.5, t = 100, m = 2.$ $\alpha = 0.7, t = 90, m = 2.$

Figure 4.1 – Complexities of the key-recovery attacks against alternant codes over $\mathbb{F}_q$ with $n = \alpha q^m$ and $m = 2$. ISD complexity (message-recovery) is indicated for completeness.

4.4.2.3 Codes With Constant Rates.

We propose another asymptotical evaluation of the complexity with parameters $n = cmt$ with c, t and base field size q constant ($c > 1$). This has the benefit of keeping a constant information rate $\frac{n-mt}{n} = 1 - \frac{1}{c}$ when m grows to infinity. With such parameters, we have

$$\log(\mathrm{R}(n,t,m,q)) = \log \binom{q^m}{cmt} + (mt - m^2 t)\log(q) - \log(t).$$

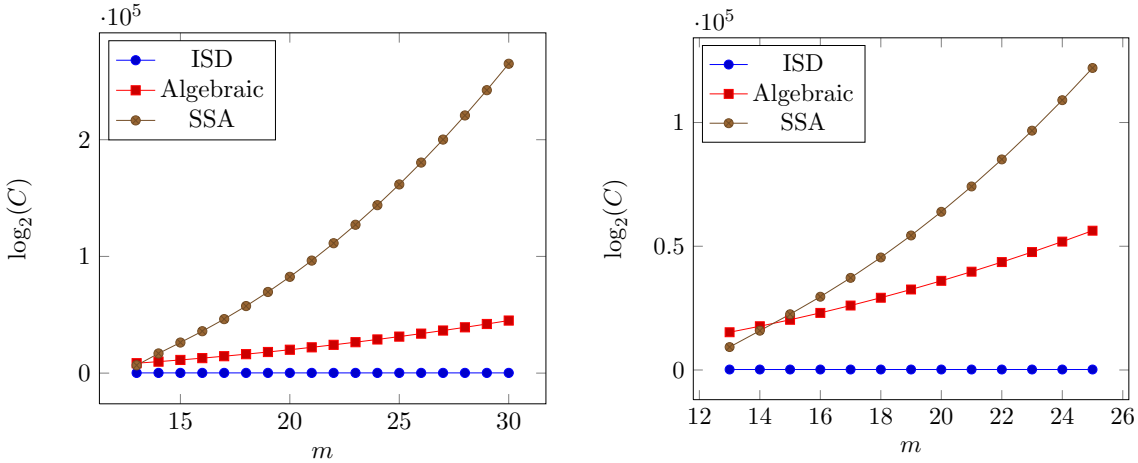
We perform the asymptotical development of $\log \binom{q^m}{cmt}$, we obtain

$$\log \binom{q^m}{cmt} = cm^2 t\log(q) + o(m^2).$$

Finally,

$$\log(\mathrm{R}(n,t,m,q)) \underset{\substack{n,m\to+\infty \\ n=cmt}}{\sim} (c-1)m^2 t\log(q) \underset{m\to+\infty}{\longrightarrow} +\infty.$$

Since $c > 1$, the FOPT attack is asymptotically better than the SSA attack. We illustrate this in Figure 4.2.



$k/n = 0.9, t = 90, q = 2.$ $k/n = 0.75, t = 90, q = 2$

Figure 4.2 – Complexities of the key-recovery attacks (algebraic attacks & SSA attack) against alternant codes over $\mathbb{F}_q$ with $n = cmt$. ISD complexity (message-recovery) is indicated for completeness.

Conclusion

We gave new algebraic modellings for very famous sub-families of alternant codes. Though they do not represent a direct threat to the corresponding McEliece scheme, their resolution gives an attack which is, for certain parameters, more efficient than the SSA attack. We insist on the fact that the bounds should be improvable by a deeper analysis of their structure. They will be a central tool for the practical attacks against alternant codes with symmetries in Chapter 6.

Chapter 5

Alternant Codes with Non-trivial Permutation Groups

This chapter is inspired by the article *Folding Alternant and Goppa Codes with Non-Trivial Automorphism Groups*, submitted to *IEEE Transactions of Information Theory*, ([FOP+14a]), written with Jean-Charles Faugère, Ayoub Otmani, Jean-Pierre Tillich and Ludovic Perret. A short version of this article was published at ISIT 2014 in [FOP+14b].

The purpose of this chapter is to detail the construction of families of codes having compact representations proposed for cryptographic use between 2005 and 2013 ([BCGO09, MB09, BCMN10, BLM11, Per12a]). Those codes all belong to the family of alternant codes (Definition 1.19) and, thanks to their compact representations, allow to design code-based encryption and signature schemes with smaller public keys. Such compact variants are the center of Chapter 6.

Section 5.1 details the advantages of alternant codes with compact representations in cryptography. We review the past proposals and explain that they share the common feature of admitting a non-trivial automorphism groups (Definition 1.13). This is why we often refer to those codes as *alternant codes with symmetries*. We give in Section 5.2 an implicit construction of GRS and alternant code with symmetries: those codes can be constructed by imposing specific relations on the support and multipliers of the alternant code (Theorem 5.4 from Dür's work [Dür87] and Theorem 5.9 from [Ber99]). We explain that there are four families of those specific relations leading to symmetric alternant codes (Theorem 5.6). In Section 5.3, we detail how to build all the past proposals directly from Theorem 5.6. We obtain in Theorem 5.10 a new description of the codes used in [MB09, BCMN10, BLM11] that are useful for the security analysis given in Chapter 6.

In Section 5.4, we show a structural weakness of all the schemes proposed for cryptography: there is in fact a smaller *hidden* Goppa (or alternant) code behind the public generator or parity-check matrix of all alternant code with symmetries. We informally summarize this below:

Fact 5.1 (informal)**.** *Let $\mathscr{C} = \mathcal{A}_t(\mathbf{x}, \mathbf{y})$ be an alternant (resp. Goppa) code of degree t, length n, with support $\mathbf{x} \in \mathbb{F}_{q^m}^n$ and multiplier $\mathbf{y} \in \mathbb{F}_{q^m}^n$ with non-trivial automorphism group as in [BCGO09, MB09, BCMN10, BLM11, Per12a]. We can construct in polynomial-time an alternant (resp. Goppa) code $\mathscr{C}' = \mathcal{A}_{t'}(\mathbf{x}', \mathbf{y}')$ of degree $t' < t$, length $n' < n$, with support $\mathbf{x}' \in \mathbb{F}_{q^m}^{n'}$ and multiplier $\mathbf{y}' \in \mathbb{F}_{q^m}^{n'}$. There is an explicit relation between $(\mathbf{x}', \mathbf{y}') \in \mathbb{F}_{q^m}^{n'} \times \mathbb{F}_{q^m}^{n'}$ and $(\mathbf{x}, \mathbf{y}) \in \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n$, that is the support/multiplier $(\mathbf{x}, \mathbf{y})$ defining $\mathscr{C}$ can be recovered from the knowledge of $(\mathbf{x}', \mathbf{y}')$.*

Typically, the length n is divided by the order of the permutation group. Roughly speaking, the new code $\mathscr{C}'$ – that we shall call *folded code* – is obtained by summing the coordinates which belong to the same orbit of the automorphism group.

This implies that the key-recovery on the McEliece variants considered in [BCGO09, MB09, BLM11, BCMN10, Per12a] is not harder than a key-recovery on a traditional McEliece cryptosystem with parameters reduced by the size of the automorphism group.

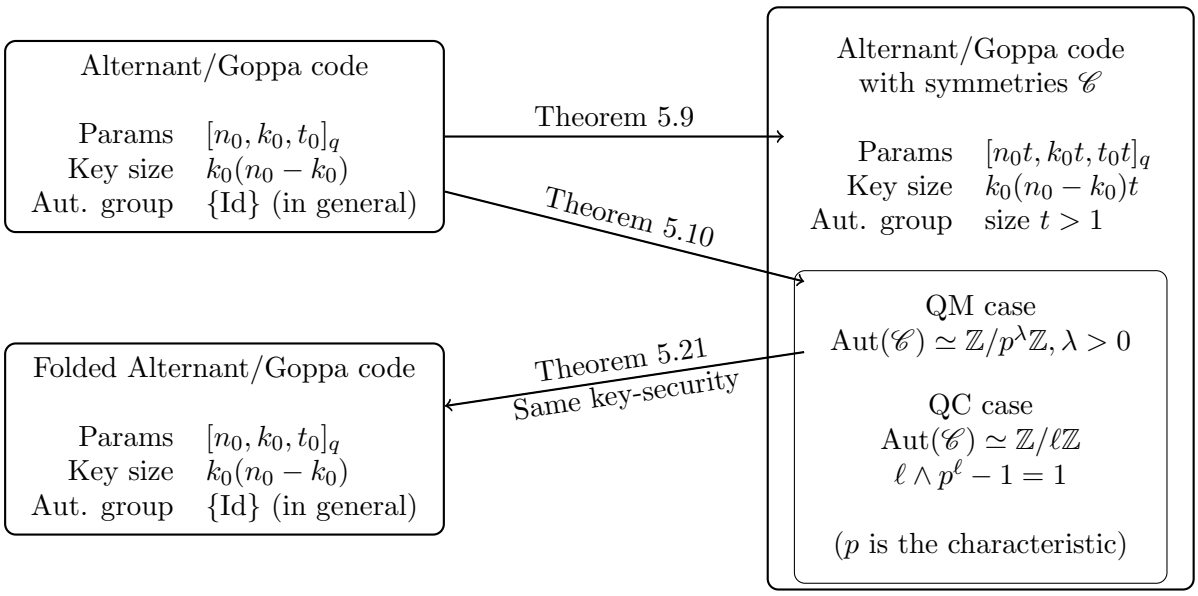


Figure 5.1 – Constructions and security of QM alternant codes.

5.1 Alternant Codes with Symmetries in Cryptography.

The motivation for using alternant codes with symmetries comes from the fact that code-based encryption and signature primitives are still not widely deployed in practice because of the sizes of public keys. Indeed, for a given security level and compared to other existing cryptosystems (RSA, elliptic curves), the sizes of the keys is at least one order of magnitude higher. For instance, to guarantee a level of security of 2^{128}, an optimized choice of the parameters for McEliece yields a key of $181,000$ bits. This figure has to be balanced with one of the most wildspread asymmetric encryption scheme, RSA, which currently provides a security factor of roughly 2^{128} with 3200-bit keys (more comparisons can be found in [NMBB12]).

The benefit of using a code with a non-trivial automorphism group is to have a more compact public key. Pick for example the code spanned by the matrix of Example 1.15,

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

The i-th row of $\mathbf{M}$ can be deduced from the first row by shifting i times on the left the coordinates, so that to know all the matrix $\mathbf{M}$ it is enough to store only the first row. More precisely, for all $0 \leqslant i, j \leqslant 4$, it holds

$$M_{i,j} = M_{0,j-i \mod 5}.$$

This property has to be combined with the existence of a trapdoor. This was the purpose of the proposals made in [BCGO09, MB09, BCMN10, BLM11], that we present now.

Quasi-Cyclic Alternant Codes

In [BCGO09], the authors build a first family of alternant codes with symmetries. They obtain a public code whose parity-check matrix $\mathbf{H}$ has a block structure:

$$\mathbf{H} = \left(\begin{array}{c|c|c} & & \\ \hline \cdots & \mathbf{M}_i & \cdots \\ \hline & & \end{array}\right) \quad \text{with } \mathbf{M}_i = \begin{pmatrix} a_0 & a_1 & \cdots & a_{\ell-1} \\ a_{\ell-1} & a_0 & \cdots & a_{\ell-2} \\ \vdots & \ddots & \ddots & \vdots \\ a_1 & a_{\ell-1} & \cdots & a_0 \end{pmatrix}.$$

Figure 5.2 – Cyclic matrices used for the blocks of the public matrix in [BCGO09].

The public keys only contain some of the entries of $\mathbf{G}$, and have sizes as low as $6,510$ bits for an ISD complexity of 2^{80}.

Quasi-Dyadic/Monoidic Alternant Codes

Some years later, it was proposed in [MB09, BCMN10, BLM11] to build Goppa codes whose generator matrix is also made of symmetric blocks, but with a form that we will refer to as *dyadic* (when the characteristic if 2, as in the example below) or *monoidic* (for other characteristics).

For a Goppa code, the authors used the fact that, if the Goppa polynomial is separable and has single roots then the code admits a parity-check matrix in *Cauchy form*

$$C(\mathbf{x},\mathbf{z}) = \left[\frac{1}{z_i - x_j}\right]_{\substack{0\leqslant i\leqslant t-1\\ 0\leqslant j\leqslant n-1}}$$

where $\mathbf{x}$ is the support of the code and $\mathbf{z}$ contains the roots of the Goppa polynomial (See [MS86][Ch. 12, §3, p.345]). Their idea is to find $\mathbf{x}$ and $\mathbf{z}$ such as $C(\mathbf{x},\mathbf{z})$ is *dyadic/monoidic.*

Definition 5.2 (Quasi-monoidic and quasi-dyadic codes [BLM11, MB09])**.** *Let $q = p^s$ (p prime, and $s > 0$). Let λ be an integer such that $p^\lambda \leqslant q$. A matrix $\mathbf{H} = (h_{i,j}) \in \mathbb{F}_q^{p^\lambda \times p^\lambda}$ is* monoidic *if $\forall i,j, 0 \leqslant i,j < p^\lambda$:*

$$h_{i,j} = h_{0,j\ominus_p i},$$

where for any integers $a \geqslant 0$ and $b \geqslant 0$ the operations $a \oplus_p b$ and $a \ominus_p b$ stand for component-wise addition and subtraction modulo p of their p-ary decomposition (we may use only $\oplus$ and $\ominus$ when p is equal to the ambient characteristic). When $p = 2$, a monoidic matrix is called a dyadic *matrix. A* quasi-monoidic *(resp.* quasi-dyadic*) matrix is a block-matrix such that each block is monoidic (resp. dyadic).*

$$\mathbf{H} = \left(\begin{array}{c|c|c} & & \\ \hline \cdots & \mathbf{M}_i & \cdots \\ \hline & & \end{array}\right) \quad \text{with } \mathbf{M}_i = \left(\begin{array}{cc|cc} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_0 & a_3 & a_2 \\ \hline a_2 & a_3 & a_0 & a_1 \\ a_3 & a_2 & a_1 & a_0 \end{array}\right).$$

Figure 5.3 – Dyadic matrices used for the blocks of the public matrix in [MB09].

In [BLM11, Theorem 1] and [MB09, Theorem 2], the authors provide a precise characterisation of the constraints that $\mathbf{x}$ and $\mathbf{z}$ should satisfy for the matrix $C(\mathbf{x},\mathbf{z})$ to be dyadic/monoidic. This leads to an algorithm [BLM11, Algo. 3] generating dyadic/monoidic parity-check matrices.

The common feature to all the alternant quasi-cyclic/dyadic/monoidic codes is to have a non-trivial automorphism group (Definition 1.13). Indeed, in all cases, the code $\mathscr{C}$ is defined by a parity-check matrix $\mathbf{H}$ and a permutation of the code position $\sigma \neq \mathrm{Id}$ such that for all the rows $\mathbf{h}$ of $\mathbf{H}$, $\mathbf{h}^\sigma$ is also a row of $\mathbf{H}$. As the rows of $\mathbf{H}$ generate the dual of $\mathscr{C}$, it is clear that σ is an automorphism of $\mathscr{C}^\perp$. Thanks to Proposition 1.14, we know that $Aut(\mathscr{C}) = Aut(\mathscr{C}^\perp)$, so that $\mathscr{C}$ has a non-trivial automorphism group (that is, not reduced to the identity).

More formally, we will deal with the following cases:

Definition 5.3. *Let $\mathscr{C}$ be an alternant or Goppa code defined over a field $\mathbb{F}_{q^m}$ ($q = p^s$ with p prime and $s > 0$) of length n. We say that $\mathscr{C}$ is:*

- *Quasi-Cyclic* (QC) *if* $\mathrm{Aut}(\mathscr{C})$ *is of the form* $(\mathbb{Z}/\ell\mathbb{Z})$ *and* $\ell \neq p$,
- *Quasi-Dyadic* (QD) *if* $p = 2$ *and* $\mathrm{Aut}(\mathscr{C})$ *is of the form* $(\mathbb{Z}/2\mathbb{Z})^\lambda$,
- *Quasi-Monoidic* (QM) *if* $\mathrm{Aut}(\mathscr{C})$ *is of the form* $(\mathbb{Z}/p\mathbb{Z})^\lambda$ *with* $p = \mathrm{char}(\mathbb{F}_{q^m}) > 2$.

A natural question emerging after these separate works is : *are there other possible symmetries for an alternant code ?* It was actually answered by Berger in [Ber99, Ber00a, Ber00b] at its most general level, building up on a work by Dür [Dür87] about the automorphism group of (generalized) Reed-Solomon codes. This is what we detail now.

5.2 Classification of Alternant Codes with Non-Trivial Permutation Groups

As alternant codes are subfield subcodes of GRS codes, we first look for GRS codes with non-trivial automorphism group (Definition 1.13).

5.2.1 GRS Codes with Non-Trivial Permutation Groups

A GRS code with non-trivial automorphism group has to satisfy for some non-trivial permutation $\sigma \in \mathcal{S}_n$ the property

$$\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})^\sigma = \mathrm{GRS}_t(\mathbf{x}, \mathbf{y}).$$

Let $\mathbf{m} \in \mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$, there exists $Q \in \mathbb{F}_{q^m}[z]$ of degree lower than t such that $\mathbf{m} = (y_0 Q(x_0), \ldots, y_{n-1} Q(x_{n-1}))$. We have

$$\mathbf{m}^\sigma = (y_0 Q(x_0), \ldots, y_{n-1} Q(x_{n-1}))^\sigma = (y_{\sigma(0)} Q(x_{\sigma(0)}), \ldots, y_{\sigma(n-1)} Q(x_{\sigma(n-1)})) \in \mathrm{GRS}_t(\mathbf{x}^\sigma, \mathbf{y}^\sigma).$$

We see that GRS codes with non-trivial automorphism can be found by looking for permutations σ such that $\mathrm{GRS}_t(\mathbf{x}^\sigma, \mathbf{y}^\sigma) = \mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$. This question was actually studied by Dür in [Dür87], which states necessary and sufficient relations on two couples $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}', \mathbf{y}')$ of support and multipliers in $\mathbb{F}_{q^m}$ such that the codes $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$ and $\mathrm{GRS}_t(\mathbf{x}', \mathbf{y}')$ over $\mathbb{F}_{q^m}$ are the same. Examples of such supports were provided by Proposition 1.20, but [Dür87] has the benefit of giving us all the possibilities of $(\mathbf{x}', \mathbf{y}')$ for a given $(\mathbf{x}, \mathbf{y})$.

Thanks to Dür's theorem (recalled here in Theorem 5.4), GRS codes with symmetries can easily be constructed by looking at the action on the support and the multipliers of the applications defined by a matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{GL}(2, \mathbb{F}_{q^m})$ by:

$$\begin{array}{rcl} \mathbb{F}_{q^m} \cup \{\infty\} & \to & \mathbb{F}_{q^m} \cup \{\infty\} \\ z & \mapsto & \dfrac{az+b}{cz+d}. \end{array}$$

However, this action on the support may transform a coordinate of the support into ∞ and a slightly more general definition of generalized Reed-Solomon codes and of alternant codes is required to cope with this issue. This construction allows to have ∞ in the support. So, in order to apply Dür 's central theorem, only in this paragraph, we pick the support and multipliers of the alternant codes in the projective line $\mathbb{P}^1(\mathbb{F}_{q^m}) = \mathbb{F}_{q^m} \cup \{\infty\}$.

The coordinates over $\mathbb{P}^1(\mathbb{F}_{q^m})$ are defined by $\phi : \mathbb{P}^1(\mathbb{F}_{q^m}) \to \mathbb{F}_{q^m}^2 \setminus \{0\}$: $\phi(\lambda) = (\lambda, 1)$ for $\lambda \in \mathbb{F}_{q^m}$ and $\phi(\infty) = (1, 0)$.

The action of $\mathrm{GL}(2, \mathbb{F}_{q^m})$ over $\mathbb{P}^1(\mathbb{F}_{q^m})$ is defined through several applications. First, we define $\Psi : \mathbb{F}_{q^m}^2 \setminus \{0\} \longrightarrow \mathbb{F}_{q^m}^* \times \mathbb{P}^1(\mathbb{F}_{q^m})$, by

$$\begin{array}{rcl} \Psi(u, v) & = & (v, u/v) \text{ if } v \neq 0, \text{ and} \\ \Psi(u, 0) & = & (u, \infty). \end{array}$$

Its inverse Φ is defined by

$$\begin{array}{rcl} \Phi(\lambda, z) & = & (\lambda z, \lambda) \text{ if } \lambda \in \mathbb{F}_{q^m}, \text{ and} \\ \Phi(\lambda, \infty) & = & (\lambda, 0). \end{array}$$

$\mathrm{GL}(2,\mathbb{F}_{q^m})$ acts on $\mathbb{F}_{q^m}^2\setminus\{0\}$ by $f(u,v)=(au+bv,cu+dv)$ for $f=\begin{pmatrix} a & b \\ c & d\end{pmatrix}\in \mathrm{GL}(2,\mathbb{F}_{q^m})$.
Denoting Ψ_1 and Ψ_2 the two components of Ψ, we obtain from f:

— an operation of $\mathrm{GL}(2,\mathbb{F}_{q^m})$ on $\mathbb{P}^1(\mathbb{F}_{q^m})$ defined by $\overline{f}(z)=\Psi_2(f(\Phi(1,z)))$. Explicitly, $\overline{f}$ acts on $\mathbb{P}^1(\mathbb{F}_{q^m})$ by

$$\overline{f}(z)=\begin{cases} \frac{az+b}{cz+d} \text{ if } z\neq\infty \text{ and } cz+d\neq 0 \\ \infty \text{ if } z=\infty \text{ and } c=0 \text{ or } cz+d=0 \\ \frac{a}{c} \text{ if } z=\infty \text{ and } c\neq 0\end{cases}.$$

A classical result on homographies shows that, for $f,g\in\mathrm{GL}(2,\mathbb{F}_{q^m})$ and $z\in\mathbb{P}^1(\mathbb{F}_{q^m})$,

$$\overline{fg}(z)=\overline{f}\circ\overline{g}(z).$$

— A function $\theta:\mathrm{GL}(2,\mathbb{F}_{q^m})\times\mathbb{P}^1(\mathbb{F}_{q^m})\longrightarrow\mathbb{F}_{q^m}^*, \theta(f,z)=\Psi_1(f(\Phi(1,z)))$. θ is given explicitly by:

$$\theta(f,z)\begin{cases} cz+d \text{ if } z\neq\infty \text{ and } cz+d\neq 0 \\ az+b \text{ if } z\neq\infty \text{ and } cz+d=0 \\ c \text{ if } z=\infty \text{ and } c\neq 0 \\ a \text{ if } z=\infty \text{ and } c=0\end{cases}.$$

θ satisfies, for $f,g\in\mathrm{GL}(2,\mathbb{F}_{q^m})$:

$$\theta(fg,z)=\theta(f,\overline{g}(z))\theta(g,z) \tag{5.1}$$

We can now state Dür's theorem in its original context:

Theorem 5.4 ([Dür87], Theorem 4)**.** *Let $(\mathbf{x},\mathbf{y}),(\mathbf{x}',\mathbf{y}')\in(\mathbb{F}_{q^m})^n\times(\mathbb{F}_{q^m}^*)^n$ and $2\leqslant k\leqslant n-2$. Then, $\mathrm{GRS}_k(\mathbf{x},\mathbf{y})=\mathrm{GRS}_k(\mathbf{x}',\mathbf{y}')$ if and only if there exist $f\in\mathrm{GL}(2,\mathbb{F}_{q^m})$ and $\lambda\in\mathbb{F}_{q^m}^*$ such that for all $0\leqslant i\leqslant n-1$,*

$$\begin{aligned} x_i' &= \overline{f}(x_i), \\ y_i' &= \lambda\theta(f,x_i)^{k-1}y_i. \end{aligned}$$

As a consequence, $\sigma\in\mathrm{Aut}(\mathrm{GRS}_k(\mathbf{x},\mathbf{y}))$ if and only if there exists $f\in\mathrm{GL}(2,\mathbb{F}_{q^m})$, $0\leqslant j\leqslant m-1$, and $\lambda\in\mathbb{F}_{q^m}^*$ such that

$$x_{\sigma(i)} = \overline{f}(x_i), \tag{5.2}$$

$$y_{\sigma(i)} = \lambda\theta(f,x_i)^{k-1}y_i. \tag{5.3}$$

This theorem gives all the possible permutations for GRS codes. When such a permutation exists, we show that there exists a new support and set of multipliers defining the same alternant code, but satisfying simpler relations. The idea is to diagonalize the matrix f. First we show that we can replace f by any matrix similar to f:

Lemma 5.5. *Let $\mathscr{C}=\mathrm{GRS}_k(\mathbf{x},\mathbf{y})$ be a code with non trivial permutation $\sigma\in\mathcal{S}_n$. Let $f\in\mathrm{GL}(2,\mathbb{F}_{q^m})$, $0\leqslant j\leqslant m-1$, and $\lambda\in\mathbb{F}_{q^m}^*$ be such that $\mathbf{x}$ and $\mathbf{y}$ satisfy (5.2) and (5.3). Let's suppose that f is similar to a matrix $M\in\mathrm{GL}(2,\mathbb{F}_{q^m})$, i.e. we can write $f=P^{-1}MP$, for some $P\in\mathrm{GL}(2,\mathbb{F}_{q^m})$. Then there exists a support $\mathbf{u}\in\left(\mathbb{P}^1(\mathbb{F}_{q^m})\right)^n$ and a set of multipliers $\mathbf{v}\in\left(\mathbb{P}^1(\mathbb{F}_{q^m})\right)^n$ such that $\mathscr{C}=\mathrm{GRS}_k(\mathbf{u},\mathbf{v})$ and*

$$\begin{aligned} u_{\sigma(i)} &= \overline{M}(u_i), \\ v_{\sigma(i)} &= \lambda\left(\theta(M,u_i)^{k-1}v_i\right). \end{aligned}$$

Proof. We know that $x_{\sigma(i)} = \left(\overline{P^{-1}MP}(x_i)\right) = \left(\overline{P^{-1}} \circ \overline{M} \circ \overline{P}(x_i)\right)$ so $\overline{P}(x_{\sigma(i)}) = \overline{M} \circ \overline{P}(x_i)$. We choose the support $\mathbf{u}$ defined by $u_i = \overline{P}(x_i)$, and $\mathbf{v}$ by $v_i = \theta(P, x_i)^{k-1} y_i$. We have $\mathscr{C} = \mathrm{GRS}_k(\mathbf{u}, \mathbf{v})$ thanks to Theorem 5.4.

Indeed, it is clear that $u_{\sigma(i)} = \overline{M}(u_i)$. For the multipliers, we obtain thanks to (5.1):

$$\begin{aligned} v_{\sigma(i)} &= \theta(P, x_{\sigma(i)})^{k-1} y_{\sigma(i)} \\ &= \underbrace{\theta(P, \overline{P^{-1}MP}(x_i))^{k-1} \lambda \theta(P^{-1}MP, x_i)^{k-1}}_{\theta(P.P^{-1}MP, x_i)^{k-1}} y_i \\ &= \lambda \theta(MP, x_i)^{k-1} y_i \\ &= \lambda \theta(M, \underbrace{\overline{P}(x_i)}_{u_i}) \underbrace{\theta(P, x_i) y_i}_{v_i} \\ &= \lambda \theta(M, u_i)^{k-1} v_i \end{aligned}$$

□

With these $\mathbf{u}, \mathbf{v}$ we can classify simply the action of σ and the possible supports and multipliers for alternant codes with symmetries.

Theorem 5.6. *Let $\mathscr{C}$ be a GRS code with a non-trivial permutation $\sigma \in \mathcal{S}_n$. Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_{q^m}^n$ be such that $\mathscr{C} = \mathrm{GRS}_k(\mathbf{x}, \mathbf{y})$, there exist $f \in \mathrm{GL}_2(\mathbb{F}_{q^m})$, and $\lambda \in \mathbb{F}_{q^m}^*$ such that one of the four following cases is satisfied:*

1. *$f = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ (with $\lambda_1 \neq \lambda_2$), and for all $0 \leqslant i \leqslant n-1$, $x_{\sigma(i)} = \frac{\lambda_1}{\lambda_2} x_i$ and $y_{\sigma(i)} = \lambda(\lambda_2)^{k-1} y_i$.*
2. *$f = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_1 \end{pmatrix}$, and for all $0 \leqslant i \leqslant n-1$, $x_{\sigma(i)} = x_i$ and $y_{\sigma(i)} = \lambda(\lambda_1)^{k-1} y_i$.*
3. *$f = \begin{pmatrix} \lambda_1 & 1 \\ 0 & \lambda_1 \end{pmatrix}$ (with $\lambda_1 \neq 0$), and for all $0 \leqslant i \leqslant n-1$, $x_{\sigma(i)} = x_i + \frac{1}{\lambda_1}$ and $y_{\sigma(i)} = \lambda(\lambda_1)^{k-1} y_i$*
4. *$f = \begin{pmatrix} 0 & -(ad-bc) \\ 1 & a+d \end{pmatrix}$ (with $\lambda_1 \neq 0$), and for all $0 \leqslant i \leqslant n-1$, $x_{\sigma(i)} = \frac{-(ad-bc)}{x_i + (a+d)}$ and $y_{\sigma(i)} = \lambda y_i (x_i + (a+d))^{k-1}$ (and $y_{\sigma(i)} = -(ad-bc)\lambda y_i$ if $x_i = -(a+d)$.*

Proof. It suffices to observe that $f \in \mathrm{GL}(2, \mathbb{F}_{q^m})$ belongs to one of the four different groups of classes of conjugacy in $\mathrm{GL}_2(\mathbb{F}_{q^m})$: $\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ (with $\lambda_1 \neq \lambda_2$), $\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_1 \end{pmatrix}$, $\begin{pmatrix} \lambda_1 & 1 \\ 0 & \lambda_1 \end{pmatrix}$, and $\begin{pmatrix} 0 & -(ad-bc) \\ 1 & a+d \end{pmatrix}$. Lemma 5.5 finishes the proof. □

5.2.2 Alternant Codes with Non-Trivial Permutation Groups

The construction of GRS codes with symmetries provided by Theorem 5.4 on GRS codes naturally yields alternant codes with same symmetries. We define them as *affine-induced* automorphisms.

Definition 5.7. *Let $2 \leqslant t \leqslant n-2$ and $\mathscr{C}$ be an alternant code over $\mathbb{F}_q$ with $(\mathbf{x}, \mathbf{y}) \in (\mathbb{F}_{q^m})^n \times (\mathbb{F}_{q^m}^*)^n$. A permutation $\sigma \in \mathcal{S}_n$ is said to be* affine-induced *if there exists $f \in \mathrm{GL}(2, \mathbb{F}_{q^m})$, $0 \leqslant j \leqslant m-1$, $\lambda \in \mathbb{F}_{q^m}^*$ and $(\mathbf{u}, \mathbf{v}) \in (\mathbb{F}_{q^m})^n \times (\mathbb{F}_{q^m}^*)^n$ such that Equations (5.2) and (5.3) are satisfied and $\mathscr{C} = \mathcal{A}_t(\mathbf{u}, \mathbf{v})$.*

With the same notations, the following proposition holds.

Proposition 5.8. *Any affine-induced permutation lies in the automorphism group of $\mathscr{C}$.*

Proof. Let $\mathscr{C}_{GRS} = \mathrm{GRS}_t(\mathbf{u}, \mathbf{v})$ and $\sigma \in \mathcal{S}_n$ such that $\mathbf{u}$ and $\mathbf{v}$ satisfy (5.2) and (5.3). Thanks to Theorem 5.4, we know that σ is a permutation of $\mathscr{C}_{GRS}$. Therefore, σ is clearly a permutation of the dual code of $\mathscr{C}_{GRS}^{\perp}$ (Proposition 1.14). Thanks to Definition 1.19, we know that $\mathcal{A}_t(\mathbf{u}, \mathbf{v})$ can be characterized as follows:

$$\mathcal{A}_t(\mathbf{u}, \mathbf{v}) = \left\{ \mathbf{m} \in \mathscr{C}_{GRS}^{\perp} \mid \forall 0 \leqslant i \leqslant n-1, m_i \in \mathbb{F}_q \right\}.$$

For $\mathbf{m} = (m_0, \ldots, m_{n-1}) \in \mathcal{A}_t(\mathbf{u}, \mathbf{v})$, $\mathbf{m}^\sigma \in \mathscr{C}_{GRS}^{\perp}$ as σ is a permutation of $\mathscr{C}_{GRS}^{\perp}$. For all $0 \leqslant i \leqslant n-1$, $m_i^\sigma = m_{\sigma(i)} \in \mathbb{F}_q$, so that $\mathbf{m}^\sigma \in \mathcal{A}_t(\mathbf{u}, \mathbf{v})$ and $\sigma \in \mathrm{Aut}(\mathcal{A}_t(\mathbf{u}, \mathbf{v}))$. □

Theorem 5.4 provides an equivalence between the existence of non-trivial automorphism and the aformentionned relations on the support and multipliers. This means that for a GRS code, no permutation σ is possible without the existence of a support and multipliers falling into one of the four cases of Theorem 5.6. This is not the case for alternant codes, as Proposition 5.7 only gives a sufficient condition. The reason is that, as an alternant codes is a subfield subcode of a GRS code, it naturally inherits the automorphisms that stabilize the corresponding GRS code, but it can be stable by other automorphisms.

In particular, it was proved by Berger that the action of homographies on the support can be combined with Fröbenius endomorphisms. This gives more families of alternant codes with non-trivial automorphism group. We will not consider them in the rest of this thesis, but we mention the following theorem for completeness.

Theorem 5.9 (Theorem 2.2,[Ber99]). *Let $2 \leqslant t \leqslant n-2$ and $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ be an alternant code over $\mathbb{F}_q$ with $(\mathbf{x}, \mathbf{y}) \in (\mathbb{F}_{q^m})^n \times (\mathbb{F}_{q^m}^*)^n$, and $\sigma \in \mathcal{S}_n$. Suppose there exists $f \in \mathrm{GL}(2, \mathbb{F}_{q^m})$, $0 \leqslant j \leqslant m-1$, and $\lambda \in \mathbb{F}_{q^m}^*$ such that:*

$$x_{\sigma(i)} = \overline{f}\,(x_i)^{q^j}, \tag{5.4}$$

$$y_{\sigma(i)} = \lambda \left(\theta\,(f, x_i)^{n-t-1}\, y_i \right)^{q^j}. \tag{5.5}$$

then σ is a permutation of $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$.

This leads Berger to adopt a larger definition of affine-induced permutations than we do here, that is, the permutations induced by relations (5.4) and (5.5). He also gives an analogous of Theorem 5.6 to alternant codes. Details can be found in from [Ber99] and [Ber00b].

Now we explain how Theorem 5.6 encompasses all the previous propositions of alternant codes with symmetries.

5.3 Link with the known constructions of alternant codes with symmetries

All the alternant codes built in [BCGO09, MB09, BCMN10, BLM11] are particular cases of Theorems 5.6 and 5.9 (with no Fröbenius action, *e.g.* $j = 0$). We detail the permutation groups obtained.

5.3.1 Quasi-Cyclic Alternant Codes ([BCGO09])

The construction proposed in [BCGO09] is exactly the case 1 of Theorem 5.6. The relations imposed on $\mathbf{x}$ and $\mathbf{y}$ by [BCGO09], are, for $a, \lambda \in \mathbb{F}_{q^m}^*$ and $0 \leqslant i \leqslant n-1$:

$$x_{\sigma(i)} = a x_i, \tag{5.6}$$
$$y_{\sigma(i)} = \lambda y_i. \tag{5.7}$$

Let ℓ be the order of σ (necessarily finite). We must have a of order ℓ otherwise $x_{\sigma^k(i)} = x_i$ for some $k < \ell$, and we have $\lambda^\ell = 1$. Then, we write $\lambda = a^e$. The automorphism group is of the form $\mathbb{Z}/\ell\mathbb{Z}$ (as in [BCGO09]), and there exists a parity-check matrix as in Figure 5.2.

5.3.2 Quasi-Dyadic/Monoidic Alternant Codes ([MB09, BCMN10, BLM11])

In [MB09, BCMN10, BLM11], the relations imposed on $\mathbf{x}, \mathbf{y}$ are, for some $b \in \mathbb{F}_{q^m}^*$ and for $0 \leqslant i \leqslant n-1$:

$$x_{\sigma(i)} = x_i + b, \tag{5.8}$$
$$y_{\sigma(i)} = y_i. \tag{5.9}$$

In case 3 of Theorem 5.6, the relations imposed on $\mathbf{x}$ are the same. For $\mathbf{y}$, we have $y_{\sigma(i)} = \lambda y_i$ but we show that necessarily $\lambda = 1$. As σ has a finite order ℓ, we know that $x_{\sigma^\ell(i)} = x_i = x_i + \ell b$. This implies that the characteristic p divides ℓ, so we write $\ell = \ell' p$ with $\ell' \geqslant 1$. We prove that $\ell' = 1$: suppose that $\ell' > 1$, then p is not the order of σ, so there exists $i_0 \in \{0, \ldots, n-1\}$ such that $\sigma^p(i_0) \neq i_0$. But it this case $x_{\sigma^p(i_0)} = x_{i_0} + pb = x_{i_0}$ with $\sigma^p(i_0) \neq i_0$: $\mathbf{x}$ cannot be a correct support (as all the elements have to be pairwise distinct). We conclude that $\ell = p$. Then for $\mathbf{y}$, the relation $y_{\sigma^\ell(i)} = y_i = \lambda^\ell y_i$ yields $\lambda^\ell = \lambda^p = 1$. So it holds that $\lambda = 1$. The consequence is that for all the dyadic and monoidic cases from [MB09, BCMN10, BLM11], the y_i's are necessarily equal on each orbit of σ. The automorphism group is of the form $\mathbb{Z}/p\mathbb{Z}$.

To build the codes that we study in Chapter 6, the authors use variations of this case leading to an automorphism group of the form $(\mathbb{Z}/p\mathbb{Z})^\lambda$. In the next paragraph, we explain in details why those codes belong to the same framework as presented here.

5.3.3 Generalization of Quasi-Monoidic Alternant Codes.

In [MB09, Algorithm 1] and [BLM11, Algorithm 3] the authors describe methods to sample efficiently QM Goppa codes. Compared to [Ber99] and Case 3 of Theorem 5.6, the codes designed in [MB09, BLM11] have the interesting property of being invariant by several permutations simultaneously, thus leading to larger permutation groups. Our Theorem 5.10 shows that those codes can be built directly by induction of Case 3 of Theorem 5.6. We complete this description by detailing the requirements on the polynomial $\Gamma(z)$ for the code $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ with $\mathbf{y} = \Gamma(\mathbf{x})^{-1}$ to be a QM Goppa code. Our construction is more convenient for Chapter 6, and allows to build the same codes as [MB09, BLM11] (actually a bit more, see Remark 5.11).

Theorem 5.10. *Let* $\mathrm{char}(\mathbb{F}_{q^m}) = p$, *and let* $\gamma(z) \in \mathbb{F}_{q^m}[z]$ *be of degree* t_0. *Let* $\alpha_0, \ldots, \alpha_{\lambda-1} \in \mathbb{F}_{q^m}$ *be a set of* λ *elements which are* $\mathbb{F}_p$*-independent over* $\mathbb{F}_{q^m}$. *We denote by* $G \subset \mathbb{F}_{q^m}$ *the additive group of the* $\mathbb{F}_p$*-linear combinations of the* α_i*'s. Let* $\widetilde{x}_0, \widetilde{x}_1, \ldots, \widetilde{x}_{(n_0-1)}$ *be elements of* $\mathbb{F}_{q^m}$ *which are in different cosets of* $\mathbb{F}_{q^m}/G$, *and* $\widetilde{y}_0, \widetilde{y}_1, \ldots, \widetilde{y}_{(n_0-1)}$ *be non-zero elements of* $\mathbb{F}_{q^m}$.

Let $(i_0, \ldots, i_{\lambda-1}) \in \mathbb{F}_p^\lambda$ *be the representation of* $i \bmod p^\lambda$ *in base* p*, i.e.* $i \equiv \sum_{j=0}^{\lambda-1} i_j p^j \bmod p^\lambda$*. Let* $n = n_0 p^\lambda$*. We define* $\mathbf{x} = (x_i)_{0 \leqslant i < n}$ *and* $\mathbf{y} = (y_i)_{0 \leqslant i < n}$ *as follows:*

$$x_i = \widetilde{x}_{\lfloor i/p^\lambda \rfloor} + \sum_{j=0}^{\lambda-1} i_j \alpha_j, \tag{5.10}$$

$$y_i = \widetilde{y}_{\lfloor i/p^\lambda \rfloor}. \tag{5.11}$$

Then, $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ *is a Goppa code of length* n *with* $t = t_0 p^\lambda$ *which admits an automorphism group of size* p^λ*.*

Moreover, we define

$$\Gamma(z) = \gamma\big(P(z)\big) \text{ where } P(z) = \textstyle\prod_{g \in G}(z - g). \tag{5.12}$$

Then, $\mathscr{G}(\mathbf{x}, \Gamma(z))$ *is a Goppa code of length* n*, degree* $t = t_0 p^\lambda$ *which admits an automorphism group of size* p^λ*. In both case, the automorphism group is given by:*

$$\mathrm{Aut}(\mathcal{A}_t(\mathbf{x}, \mathbf{y})) = \mathrm{Aut}(\mathscr{G}(\mathbf{x}, \Gamma(z))) \supseteq \left\{ \sigma_\ell \,\middle|\, i \in \{0, \ldots, n-1\} \mapsto i \ominus_p \ell, 0 \leqslant \ell \leqslant p^\lambda - 1 \right\},$$

where $\oplus_p$ *is as in Definition 5.2.*

Proof. Let us show that all the permutations $\sigma_\ell(i) = i \ominus \ell$, for $\ell, 0 \leqslant \ell \leqslant p^\lambda - 1$ are code permutations. We start by proving it for $\ell = p^u$, with $u, 0 \leqslant u \leqslant \lambda - 1$. For $i, 0 \leqslant i \leqslant t_0 p^\lambda - 1$, let $(i_0, \ldots, i_{\lambda-1}) \in \mathbb{F}_p^\lambda$ be such that $i \equiv \sum_{j=0}^{\lambda-1} i_j p^j \bmod p^\lambda$. The representation $(i'_0, \ldots, i'_{\lambda-1})$ of $(i \ominus p^u) \bmod p^\lambda$ in base p is deduced from the previous representation by

$$\begin{aligned} i'_j &= i_j, \quad \text{if } j \neq u, \\ i'_u &\equiv i_u - 1 \bmod p \text{ , otherwise.} \end{aligned}$$

Hence, $i \ominus p^u$ and i only differ in their u-th digit. Since u is smaller than λ, we get:

$$\lfloor (i \ominus p^u)/p^\lambda \rfloor = \lfloor i/p^\lambda \rfloor.$$

This yields immediatly

$$y_{\sigma_{p^u}(i)} = \widetilde{y}_{\lfloor i/p^\lambda \rfloor} = y_i.$$

Moreover, thanks to Equation (5.10) we can write:

$$\begin{aligned} x_{i \ominus p^u} &= \widetilde{x}_{\lfloor (i \ominus p^u)/p^\lambda \rfloor} + \sum_{j=0}^{\lambda-1} i'_j \alpha_j = \widetilde{x}_{\lfloor i/p^\lambda \rfloor} + \alpha_u + \sum_{j=0}^{\lambda-1} i_j \alpha_j \\ &= x_i + \alpha_u. \end{aligned} \tag{5.13}$$

Now pick any ℓ in $[0, \ldots, p^\lambda - 1]$ and decompose it in base p: $\ell = \sum_{j=0}^{\lambda-1} \ell_j p^j$. A quick induction on Equation (5.13) shows that for all $i, 0 \leqslant i \leqslant t_0 p^\lambda - 1$:

$$\begin{aligned} x_{i \ominus \ell} &= x_i + \textstyle\sum_{j=0}^{\lambda-1} \ell_j \alpha_j \\ &= x_i + g_\ell \end{aligned}$$

where $g_\ell = \sum_{j=0}^{\lambda-1} \ell_j \alpha_j \in G$. Recall that G is the group of all the $\mathbb{F}_p$ linear combinations of the α_i's.

This proves that $\mathbf{x}^{\sigma_\ell}, \mathbf{x}, \mathbf{y}^{\sigma_\ell}$ and $\mathbf{y}$ satisfy the relations of Case 3 of Theorem 5.6 (with $1/\lambda_1 = g_\ell$) which are particular case of Dür's Theorem 5.4. Therefore $\sigma_\ell \in \mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$ and *a fortiori* $\sigma_\ell \in \mathcal{A}_t(\mathbf{x}, \mathbf{y})$ for all $0 \leqslant \ell \leqslant p^\lambda - 1$. Hence, $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ admits an automorphism group of cardinality at least p^λ.

Concerning the Goppa polynomial of (5.12), for all $\ell \in [0, \ldots, p^\lambda - 1]$, and $g_\ell \in G$ we have

$$\Gamma(z + g_\ell) = \gamma \left(\prod_{g \in G} (z + g_\ell - g) \right) = \gamma \left(\prod_{g \in G} (z - g) \right) = \Gamma(z).$$

This is due to the fact that G is a group. As a consequence, the multipliers associated to $\mathscr{G}(\mathbf{x}, \Gamma(z))$ satisfy for all $0 \leqslant \ell \leqslant p^\lambda - 1$:

$$y_{\sigma_\ell(i)} = \frac{1}{\Gamma(x_{\sigma_\ell(i)})} = \frac{1}{\Gamma(x_i + g_\ell)} = \frac{1}{\Gamma(x_i)} = y_i.$$

The conditions of Case 3 of Theorem 5.6 are satisfied for all σ_ℓ with $0 \leqslant \ell \leqslant p^\lambda - 1$.

□

Remark 5.11. *An example of resolution of* (5.10) *is in [MB09, Algorithm* 1*] and [BLM11, Algorithm* 3*] (by setting* $\alpha_i = h_{a_i}^{-1} + \omega$ *with the notations of [BLM11]). However, those consider only polynomials* $\gamma(z)$ *of degree* $t_0 = 1$*. Thus, we can obtain from Theorem 5.10 more codes than with the constructions provided in [MB09, BLM11].*

An Example of QM **Goppa code.** We present a detailed example of how to use Theorem 5.10 to build a Goppa code with automorphism group $(\mathbb{Z}/2\mathbb{Z})^2$.

Example 5.12. *We work in* $\mathbb{F}_{2^5} = \frac{\mathbb{F}_2[\omega]}{(\omega^5+\omega^2+1)}$*. Let us start by considering a very simple Goppa code of length* $n_0 = 7$ *and order* $t_0 = 1$*. The code is defined by its Goppa polynomial* $\Gamma_0(z) = z + \omega^{10}$ *and its support* $\widetilde{\mathbf{x}}^{(0)}$*, from which its multipliers* $\widetilde{\mathbf{y}}^{(0)}$ *are deduced:*

$$\begin{array}{rcl} \widetilde{\mathbf{x}}^{(0)} & = & (0, \omega^{26}, \omega^8, \omega^{14}, \omega^{16}, \omega^{11}, \omega^{27}) \\ \widetilde{\mathbf{y}}^{(0)} & = & (\omega^{21}, \omega^{12}, \omega^{18}, \omega^{11}, \omega^{25}, \omega^3, \omega^{22}). \end{array}$$

We construct the code $\mathscr{C}_0 = \mathscr{A}_{t_0}(\widetilde{\mathbf{x}}^{(0)}, \widetilde{\mathbf{y}}^{(0)})$*. Its generator matrix* $G^{(0)}$ *is*

$$G^{(0)} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

First symmetry. *To introduce a first symmetry, we build from* $\widetilde{\mathbf{x}}^{(0)}$ *a support* $\widetilde{\mathbf{x}}^{(1)}$ *satisfying* (5.10) *(with* $\lambda = 1$*). To do so, we pick* $\widetilde{x}^{(1)}_{2i} = \widetilde{x}^{(0)}_i$ *and apply* $\widetilde{x}^{(1)}_{i \ominus 1} = \widetilde{x}^{(1)}_i + \alpha_1$ *with* $\alpha_1 = 1$*. This gives* $\widetilde{\mathbf{x}}^{(1)} = (0, 1, \omega^{26}, \omega^{28}, \omega^8, \omega^{20}, \omega^{14}, \omega^{13}, \omega^{16}, \omega^9, \omega^{11}, \omega^{19}, \omega^{27}, \omega^6)$*. We pick a Goppa polynomial such that the multipliers are constant over each block of size* $t_1 = 2$*:* $\Gamma_1(z) = z(z + \alpha_1) + \omega^3 = z^2 + z + \omega^3$*. The corresponding multipliers are:* $\widetilde{\mathbf{y}}^{(1)} = (\omega^{28}, \omega^{28}, \omega^{20}, \omega^{20}, \omega^7, \omega^7, \omega^{13}, \omega^{13}, \omega^{21}, \omega^{21}, \omega^{22}, \omega^{22}, \omega^{11}, \omega^{11})$*. Then, the code* $\mathscr{C}_1 = \mathscr{A}_{t_1}(\widetilde{\mathbf{x}}^{(1)}, \widetilde{\mathbf{y}}^{(1)}) = \mathscr{G}(\widetilde{\mathbf{x}}^{(1)}, \Gamma_1(z))$ *admits as generator matrix:*

$$G^{(1)} = \left(\begin{array}{cc|cc|cc|cc|cc|cc|cc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{array} \right).$$

In each square block, the second row deduces from the first one by the relation $g_{1,j} = g_{0,j \ominus 1}$*.*

Second symmetry. *We build a support* $\widetilde{\mathbf{x}}^{(2)}$ *satisfying* $\widetilde{x}^{(2)}_{i\ominus 1} = \widetilde{x}^{(2)}_i + \alpha_0$ *with* $\alpha_0 = \omega^{17}$. *From the first symmetry it inherits the relation* $\widetilde{x}^{(2)}_{i\ominus 2} = \widetilde{x}^{(2)}_i + \alpha_1$. *This yields*
$\widetilde{\mathbf{x}}^{(2)} = (0, \omega^{17}, 1, \omega^{30}, \omega^{26}, \omega^2, \omega^{28}, \omega^5, \omega^8, \omega^{24}, \omega^{20}, \omega^{15}, \omega^{14}, \omega^{12}, \omega^{13}, \omega^{23}, \omega^{16}, \omega^3, \omega^9, \omega^{29}, \omega^{11}, \omega^7, \omega^{19}, \omega^{22}, \omega^{27}, \omega^{21}, \omega^6, \omega^{25})$
We pick a Goppa polynomial such that the multipliers are constant over each block of size $t_2 = 4$: $\Gamma_2(z) = z(z+\alpha_1)(z+\alpha_0)(z+\alpha_0+\alpha_1) + \omega^2 = z^4 + \omega^9 z^2 + \omega^{16} z + \omega^2$. *The corresponding multipliers are:* $\widetilde{\mathbf{y}}^{(2)} = (\omega^{29}, \omega^{29}, \omega^{29}, \omega^{29}, \omega^3, \omega^3, \omega^3, \omega^3, 1, 1, 1, 1, \omega^{26}, \omega^{26}, \omega^{26}, \omega^{26}, \omega^{14}, \omega^{14}, \omega^{14}, \omega^{14}, \omega, \omega, \omega, \omega, \omega^5, \omega^5, \omega^5, \omega^5)$.
Then, the code $\mathscr{C}_2 = \mathscr{A}_{t_2}(\widetilde{\mathbf{x}}^{(2)}, \widetilde{\mathbf{y}}^{(2)}) = \mathscr{G}(\widetilde{\mathbf{x}},^{(2)}, \Gamma_2(z))$ *admits as generator matrix* $G^{(2)}$:

$$G^{(2)} = \left(\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc}
1&0&0&0&0&0&0&0&1&0&1&1&0&0&0&1&0&1&1&1&0&0&0&1&1&1&1&1\\
0&1&0&0&0&0&0&0&0&1&1&1&0&0&1&0&1&0&1&1&0&0&1&0&1&1&1&1\\
0&0&1&0&0&0&0&0&1&1&1&0&0&1&0&0&1&1&0&1&0&1&0&0&1&1&1&1\\
0&0&0&1&0&0&0&0&1&1&0&1&1&0&0&0&1&1&1&0&1&0&0&0&1&1&1&1\\
\hline
0&0&0&0&1&0&0&0&0&1&1&0&1&1&1&0&0&1&0&0&0&1&1&1&0&1&0&1\\
0&0&0&0&0&1&0&0&1&0&0&1&1&1&0&1&1&0&0&0&1&0&1&1&1&0&1&0\\
0&0&0&0&0&0&1&0&1&0&0&1&1&0&1&1&0&0&0&1&1&1&0&1&0&1&0&1\\
0&0&0&0&0&0&0&1&0&1&1&0&0&1&1&1&0&0&1&0&1&1&1&0&1&0&1&0
\end{array}\right)$$

Note that each square block deduces from its first row thanks to the relation $g_{4i_0+i,j} = g_{4i_0,i\ominus j}$ for $0 \leqslant i_0 \leqslant 1, 0 \leqslant i \leqslant t_2 - 1, 0 \leqslant j \leqslant n_2 - 1$.

5.4 The Folded Code

Alternant codes and Goppa codes in particular with a certain non-trivial automorphism group (as considered in Section 5.3) meet a very peculiar property. Namely it is possible to derive a new alternant (or a Goppa code) with smaller parameters by simply summing up the coordinates. To define this new code more precisely, we introduce the following operator.

Definition 5.13 (Folded code)**.** *Let* $\mathscr{C}$ *be a code and* $\sigma \in \mathrm{Aut}(\mathscr{C})$ *of order* ℓ. *For each orbit of* σ *we choose one representative (for instance the smallest one). Let* $i_0, i_1, \ldots, i_{s-1}$ *be the set of these representatives. The* folded code *of* $\mathscr{C}$ *with respect to* σ, *denoted by* $\overline{\mathscr{C}^\sigma}$, *is a code of length* s *which is given by the set of words* $\overline{\mathbf{c}}^\sigma = \left(\sum_{u=0}^{\ell-1} c_{\sigma^u(i_j)}\right)_{0\leqslant j\leqslant s-1}$, *where* $\mathbf{c}$ *ranges over* $\mathscr{C}$.

Remark 5.14. *This folded code is related to constructions which were considered in the framework of decoding codes with non-trivial automorphism group [Leg11, Leg12]. We use it for a different purpose here.*

Under mild assumptions, it can be shown that the dimension of the folded code gets reduced by the order of σ. More precisely:

Proposition 5.15. *Let* $\mathscr{C}$ *be a* $[n,k]$ *code of and* $\sigma \in \mathrm{Aut}(\mathscr{C})$ *of order* ℓ. *Let* $\mathbf{G}$ *be a generator matrix with rows* $\{\mathbf{g}_0, \ldots, \mathbf{g}_{k-1}\}$. *Assume that* $\{\mathbf{g}_0, \ldots, \mathbf{g}_{k-1}\}$ *is the union of orbits of size* ℓ *under the action of* σ *defined by* $\mathbf{g}_i \mapsto \mathbf{g}_i^\sigma$.

Then, the dimension of $\overline{\mathscr{C}^\sigma}$ *is equal to* $\frac{k}{\ell}$.

Proof. This follows from the fact that $\overline{\mathscr{C}^\sigma}$ is generated by the set of $\widetilde{\mathbf{g}_{i_j}} = \sum_{u=0}^{\ell-1} \mathbf{g}_{i_j}^{\sigma^u}$ where the $\mathbf{g}_i$'s are k/ℓ representatives of each orbit of σ acting on $\{\mathbf{g}_0, \ldots, \mathbf{g}_{k-1}\}$. The $\widetilde{\mathbf{g}_{i_j}}$'s are

independent. Indeed, since for $\lambda_{i_0}, \ldots, \lambda_{i_{k/\ell-1}} \in \mathbb{F}_q$,

$$\sum_{0 \leqslant j \leqslant k/\ell-1} \lambda_{i_j} \widetilde{\mathbf{g}_{i_j}} = \sum_{\substack{0 \leqslant j \leqslant k/\ell-1 \\ 0 \leqslant u \leqslant \ell-1}} \lambda_{i_j} \mathbf{g}_{i_j}^{\sigma^u},$$

any non-trivial vanishing linear combination of the $\widetilde{\mathbf{g}_{i_j}}$ immediately yields a non-trivial vanishing linear combination of the $\mathbf{g}_i$'s. □

Remark 5.16. *A generator matrix of this form is precisely what is achieved by all the constructions of monoidic alternant/Goppa/Srivastava codes proposed in [BCGO09, MB09, BCMN10, BLM11, Per12a].*

5.5 Folding QC and QM GRS Codes

If we consider the monoidic alternant or Goppa codes constructed in [BCGO09, MB09, BLM11, BCMN10] they have typically length of the form $n = n_0\ell$, degree of the form $r = r_0\ell$ and dimension of the form $k = n - rm = \ell(n_0 - r_0 m)$ where m is the extension degree of the alternant/Goppa code and ℓ is the order of the affine-induced automorphism of the code. The automorphism group of these codes satisfies the assumptions of Proposition 5.15 and therefore the folded code has length n_0 and dimension $n_0 - r_0 m$. We introduce and study the so-called *affine-invariant* polynomials that appear when folding QC and QM alternant codes with symmetries.

5.5.1 Affine-Invariant Polynomials

Let $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ be an alternant code with a non-trivial affine-induced permutation σ (Proposition 5.7). As in Definition 1.19, we set for $0 \leqslant i \leqslant n-1$

$$y'_i = \frac{1}{y_i} \frac{1}{\prod\limits_{\substack{0 \leqslant j < n \\ j \neq i}} (x_j - x_i)},$$

so that $\mathcal{A}_t(\mathbf{x}, \mathbf{y})$ is the subfield subcode of $\mathrm{GRS}_{n-t}(\mathbf{x}, \mathbf{y}')$. As σ is affine-induced, it is also an automorphism of $\mathrm{GRS}_{n-t-1}(\mathbf{x}, \mathbf{y}')$, and we know that there exist a support $\mathbf{u}$ and multipliers $\mathbf{v}$ satisfying relations of type (5.4) and (5.5) and such that $\mathrm{GRS}_{n-t}(\mathbf{x}, \mathbf{y}') = \mathrm{GRS}_{n-t}(\mathbf{u}, \mathbf{v})$.

The explicit form of the coordinates of $\mathbf{m} = (m_0, \ldots, m_{n-1}) \in \mathcal{A}_t(\mathbf{x}, \mathbf{y}) = \mathrm{GRS}_{n-t}(\mathbf{u}, \mathbf{v})_{|\mathbb{F}_q}$ is given by

$$\mathbf{m} = (v_0 Q(u_0), \ldots, v_{n-1} Q(u_{n-1}))$$

with $Q \in \mathbb{F}_{q^m}[z]_{\leqslant n-t-1}$ such that $\forall 0 \leqslant i \leqslant n-1, v_i Q(u_i) \in \mathbb{F}_q$.

Let ℓ be the order of σ. A codeword $\mathbf{m}' = (m'_{i_0}, \ldots, m'_{i_{n/\ell-1}}) \in \overline{\mathcal{A}_t(\mathbf{x}, \mathbf{y})^\sigma}$ is as follows

$$m'_{i_j} = v_{i_j} Q(u_{i_j}) + v_{\sigma(i_j)} Q(u_{\sigma(i_j)}) \cdots + v_{\sigma^{\ell-1}(i_j)} Q(u_{\sigma^{\ell-1}(i_j)}).$$

For instance in the QC case, there exist $a, \lambda \in \mathbb{F}_{q^m}^*$ such that $\mathbf{u}$ and $\mathbf{v}$ satisfy relations (5.6) and (5.7). We have then

$$\begin{aligned} m'_{i_j} &= v_{i_j} Q(u_{i_j}) + v_{\sigma(i_j)} Q(u_{\sigma(i_j)}) \cdots + v_{\sigma^{\ell-1}(i_j)} Q(u_{\sigma^{\ell-1}(i_j)}) \\ &= v_{i_j} Q(u_{i_j}) + \lambda v_{i_j} Q(a u_{i_j}) + \cdots + \lambda^{\ell-1} v_{i_j} Q(a^{\ell-1} u_{i_j}) \\ &= v_{i_j} (Q(u_{i_j}) + \lambda Q(a u_{i_j}) + \cdots + \lambda^{\ell-1} Q(a^{\ell-1} u_{i_j})) \\ &= v_{i_j} P(u_{i_j}), \end{aligned}$$

with $P(z) = Q(z) + \lambda Q(az) + \cdots + \lambda^{\ell-1} Q(a^{\ell-1} z)$.

For a quasi-monoidic alternant code, the support and multipliers satisfy relations (5.8) and (5.9). The folded code's coordinates are

$$\begin{aligned} m'_{i_j} &= v_i Q(u_{i_j}) + v_{\sigma(i_j)} Q(u_{\sigma(i_j)}) \cdots + v_{\sigma^{\ell-1}(i_j)} Q(u_{\sigma^{\ell-1}(i_j)}) \\ &= v_{i_j}(Q(u_{i_j}) + Q(u_{i_j} + b) + \cdots + Q(u_{i_j} + (p-1)b)) \\ &= v_{i_j} P(u_{i_j}), \end{aligned}$$

with $P(z) = Q(z) + Q(z+b) + \cdots + Q(z + (p-1)b)$.

In both case, the form $v_{i_j} P(u_{i_j})$ of the folded code's coordinates shows that these words are elements of an alternant code. To understand more in details their structure and show that they are indeed alternant codes, we need to study those sums of polynomials (P in the examples above). We gather them under the following form:

$$P(z) = \sum_{i=0}^{\ell-1} \lambda^i Q(\phi^i(z)) \tag{5.14}$$

where Q is a polynomial, ϕ an affine map of order ℓ and λ an ℓ-th root of unity.

We show here a fundamental result on the form taken by those polynomials. First, such polynomial are said to be *affine-invariant*, that is:

$$\begin{aligned} P(\phi(z)) &= \sum_{i=0}^{\ell-1} \lambda^i Q(\phi^{i+1}(z)) = \frac{1}{\lambda} \sum_{i=0}^{\ell-1} \lambda^{i+1} Q(\phi^{i+1}(z)) \\ &= \frac{1}{\lambda} \left(\sum_{i=1}^{\ell-1} \lambda^i Q(\phi^i(z)) + \lambda^\ell Q(\phi^\ell(z)) \right) = \frac{1}{\lambda} \sum_{i=0}^{\ell-1} \lambda^i Q(\phi^i(z)) \\ &= \frac{1}{\lambda} P(z). \end{aligned}$$

So, there exists an ℓ-th root of unity α in $\mathbb{F}_{q^m}$ which is such that P satisfies the central equation:

$$P(az + b) = \alpha P(z). \tag{5.15}$$

Satisfying Equation (5.15) is not an insignificant property. For instance, it is well known that, working in a field $\mathbb{F}$ of characteristic different from 2, a polynomial P satisfies for any $z, P(z) = P(-z)$, if and only if there exist polynomials Q and R such that

$$P(z) = R(z^2) = Q(z) + Q(-z).$$

This means that, for any $Q \in \mathbb{F}[z]$ the sum $Q(z) + Q(\phi(z))$ with $\phi(z) = -z$ can be expressed as the composition of a polynomial R in smaller degree (as $\deg(R) \leqslant \lfloor \deg(Q)/2 \rfloor$) and the polynomial $z \mapsto z^2$.

Propositions 5.17 and 5.18 generalize this result to the polynomials satisfying Equation (5.15) for any affine map $\phi : z \mapsto az + b$. First, Proposition 5.17 characterizes all solutions of the polynomial Equation (5.15). The proof of this proposition is technical and is reported to the end of this chapter (Paragraph 5.7.1).

Proposition 5.17. *Let $\mathbb{F}_{q^m}$ be a field of finite characteristic p and let a, b, α be elements of $\mathbb{F}_{q^m}$ such that (i) $a \neq 0$ and (ii) $b \neq 0$ when $a = 1$. Let ϕ be the affine map $\phi : z \to az + b$ of order ℓ. All the polynomials $P(z) \in \mathbb{F}_{q^m}[z]$ satisfying $P(az + b) = \alpha P(z)$ with α and ℓ-root of unity have the following form*

— *If $a = 1$ then necessarily $\alpha = 1$, $\ell = p$ and there exists $R \in \mathbb{F}_{q^m}[z]$ such that*

$$P(z) = R(z^p - b^{p-1}z),$$

and $P(z)$ is of degree a multiple of p.

— *If $a \neq 1$, let z_0 be the unique fixed point of ϕ. Then, there exists d a unique integer in the range $[0, \ldots, \ell - 1]$ such that $\alpha = a^d$ and there exists $R \in \mathbb{F}_{q^m}[z]$ such that*

$$P(z) = (z - z_0)^d R\left((z - z_0)^\ell\right),$$

and $P(z)$ is of degree equal to d modulo ℓ.

Conversely, and this is crucial in our context, it turns out that all these solutions are sums of polynomials as in (5.14).

Proposition 5.18. *Let $\mathbb{F}_{q^m}$ be a finite field of characteristic p. Let a and b be as in Proposition 5.17, ϕ be the affine map $z \mapsto az + b$, λ an element of $\mathbb{F}_{q^m}$ with $\lambda = 1$ if $a = 1$, and $\lambda = a^d$ for some d in $[0, \ldots, \ell - 1]$ if $a \neq 1$ of order ℓ. We define S by*

$$\begin{array}{rcl} S : \mathbb{F}_{q^m}[z] & \to & \mathbb{F}_{q^m}[z] \\ Q(z) & \mapsto & \displaystyle\sum_{i=0}^{\ell-1} \lambda^i Q(\phi^i(z)). \end{array}$$

Then, it holds that:

— *If $a = 1$, we have for every nonnegative integer t:*

$$S\left(\mathbb{F}_{q^m}[z]_{\leqslant t}\right) = \left\{ R(z^p - b^{p-1}z) \mid \deg R \leqslant \left\lfloor \frac{t - p + 1}{p} \right\rfloor \right\}. \tag{5.16}$$

— *If $a \neq 1$, let z_0 be the unique fixed point of ϕ, and we have:*

$$S\left(\mathbb{F}_{q^m}[z]_{\leqslant t}\right) = \left\{ (z - z_0)^{\ell - d} R((z - z_0)^\ell) \mid \deg R \leqslant \left\lfloor \frac{t - \ell + d}{\ell} \right\rfloor \right\}. \tag{5.17}$$

The proof can be found in Section 5.7, Paragraph 5.7.2. Let's use these results to study the structure of the folded codes of QC and QM alternant codes.

5.5.2 Folding QC and QM GRS Codes

We start by using Propositions 5.17 and 5.18 to describe precisely the structure of the folded of a GRS code with non-trivial automorphism. Consider a QC or QM GRS code $\mathscr{C}$ over $\mathbb{F}_q$ of length n. Let $\sigma \in \mathrm{Aut}(\mathscr{C})$ be induced by the affine map $\phi : z \to az + b$ where $a, b \in \mathbb{F}_{q^m}$ are such that $a \neq 0$ and $b \neq 0$ when $a = 1$. There exist a support $\mathbf{x} \in \mathbb{F}_{q^m}^n$, multiplier $\mathbf{y} \in \mathbb{F}_{q^m}^n$ and $\lambda \in \mathbb{F}_{q^m}$ an ℓ-th root of unity such that, for all $i \in \{0, 1, \ldots, n-1\}$, $x_{\sigma(i)} = \phi(x_i)$, $y_{\sigma(i)} = \lambda y_i$ and $\mathscr{C} = \mathrm{GRS}_k(\mathbf{x}, \mathbf{y})$. We show that the folded code of $\mathscr{C}$ is a GRS code and give an explicit link between its support and multipliers and $\mathbf{x}, \mathbf{y}$.

Theorem 5.19. *We use the notations just introduced. Let ℓ be the order of σ. We denote by d the integer in $\{0, 1, \ldots, \ell - 1\}$ verifying $\lambda = a^d$. We assume that the support does not contain the fixed point of ϕ, then the action of σ on $\{0, 1, \ldots, n-1\}$ has $\frac{n}{\ell}$ orbits, each of them being of size ℓ. Choose a representative $i_0, i_1, \ldots, i_{n/\ell - 1}$ in each of these orbits.*

Then, there exists $\mathbf{x}', \mathbf{y}' \in \mathbb{F}_{q^m}^{n/\ell}$ *and a integer* r *such that*

$$\overline{\mathscr{C}^\sigma} = \overline{(\mathrm{GRS}_k(\mathbf{x}, \mathbf{y}))^\sigma} = \mathrm{GRS}_r(\mathbf{x}', \mathbf{y}')$$

with:

— *when* $a = 1$*, then* $r = \left\lfloor \frac{k-\ell}{\ell} \right\rfloor + 1$ *and for all* $j \in \{0, \dots, n/\ell - 1\}$*:*

$$x'_j = x_{i_j}^\ell - b^{\ell-1} x_{i_j} \qquad \text{and} \qquad y'_j = y_{i_j},$$

— *when* $a \neq 1$*, let us denote by* z_0 *the unique fixed point in* $\mathbb{F}_{q^m}$ *of* ϕ*. Then* $r = \left\lfloor \frac{k-\ell+d-1}{\ell} \right\rfloor + 1$ *and for all* $j \in \{0, \dots, n/\ell - 1\}$*:*

$$x'_j = (x_{i_j} - z_0)^\ell \qquad \text{and} \qquad y'_j = y_{i_j}(x_{i_j} - z_0)^{\ell-d}.$$

Proof. We divide the proof in two parts.

The case $a = 1$**.** We recall that, from Paragraph 5.3.2, $\ell = p$ and that is constant over each orbit $\{i, \sigma(i), \dots, \sigma^{\ell-1}(i)\}$. From Definition 1.17,

$$\mathscr{C} = \left\{ \left(y_i Q(x_i)\right)_{0 \leqslant i < n} \mid Q \in \mathbb{F}_{q^m}[z], \deg Q \leqslant k-1 \right\}.$$

The folded code of $\mathscr{C}$ can now be described as in the examples of Paragraph 5.5.1:

$$\overline{\mathscr{C}^\sigma} = \left\{ \left(y_{i_j} \sum_{s=0}^{\ell-1} Q\left(\phi^s(x_{i_j})\right) \right)_{j=0}^{n/\ell-1} \mid Q \in \mathbb{F}_{q^m}[z], \deg Q \leqslant k-1 \right\}$$

where $x_{i_0}, x_{i_1}, \dots, x_{n/\ell-1}$ are representatives of each of the n/ℓ orbits $\{u, \sigma(u), \dots, \sigma^{\ell-1}(u)\}$ (they have all the same size ℓ).

By using Proposition 5.18, we obtain:

$$\overline{\mathscr{C}^\sigma} = \left\{ \left(y_{i_j} R\left(x_{i_j}^p - b^{p-1} x_{i_j} \right) \right)_{j=0}^{n/\ell-1} \mid R \in \mathbb{F}_{q^m}[z], \deg R \leqslant \left\lfloor \frac{k-1-p}{p} \right\rfloor \right\}.$$

In other words, from Definition 1.17, it holds that

$$\overline{\mathscr{C}^\sigma} = \mathrm{GRS}_r(\mathbf{x}', \mathbf{y}')$$

with $r = \left\lfloor \frac{k-1-p}{p} \right\rfloor + 1$ and for any $j \in \{0, 1, \dots, n/\ell - 1\}$, $x'_j = x_{i_j}^p - b^{p-1} x_{i_j}$ and $y'_j = y_{i_j}$.

The case $a \neq 1$**.** The difference with the previous situation lies in the fact that now the y_j's are not necessarily constant over an orbit. As previously, we consider representatives $x_{i_0}, x_{i_1}, \dots, x_{n/\ell-1}$ of the n/ℓ orbits $\{u, \sigma(u), \dots, \sigma^{\ell-1}(u)\}$ (they have all the same size ℓ because the support $\mathbf{x}$ does not contain the fixed point of ϕ). We obtain that the folded code of $\mathscr{C}$ can now be described as follows.

$$\overline{\mathscr{C}^\sigma} = \left\{ \left(\sum_{s=0}^{\ell-1} y_{i_j} \lambda^s P\left(\phi^s(x_{i_j})\right) \right)_{j=0}^{n/\ell-1} \mid P \in \mathbb{F}_{q^m}[z], \deg P \leqslant k-1 \right\}.$$

We necessarily have $\lambda^\ell = 1$. Since a is a primitive ℓ-root of unity, there exists an integer d in $\{0, \dots, \ell-1\}$ such that $\lambda = a^d$. We introduce the fixed point z_0 of ϕ, and we can apply Proposition 5.18, we deduce that:

$$\overline{\mathscr{C}^\sigma} = \left\{ \left(y_{i_j}(x_{i_j} - z_0)^{\ell-d} R((x_{i_j} - z_0)^\ell) \right)_{j=0}^{n/\ell-1} \mid R \in \mathbb{F}_{q^m}[z], \deg R \leqslant \left\lfloor \frac{k-1-\ell+d}{\ell} \right\rfloor \right\}.$$

Finally, by Lemma 1.21 again we see that

$$\overline{\mathscr{C}^\sigma} = \mathrm{GRS}_r(\mathbf{x}', \mathbf{y}')$$

where $r = \lfloor \frac{k-1-\ell+d}{\ell} \rfloor + 1$, $x'_j = (x_{i_j} - z_0)^\ell$ and $y'_j = y_{i_j}(x_{i_j} - z_0)^{\ell-d}$ for any $j \in \{0, 1, \dots, n/\ell - 1\}$. □

5.6 Folding QC and QM Alternant and Goppa codes

Thanks to Theorem 5.19, we prove easily that the folded code of an alternant code is *included* in an alternant code. Indeed, consider an alternant code $\mathscr{C} = \mathcal{A}_t(\mathbf{x}, \mathbf{y})$ defined over $\mathbb{F}_q$ with affine-induced automorphism σ of order ℓ associated to the affine map ϕ acting on the support. It is the subfield subcode of $\mathrm{GRS}_{n-t}(\mathbf{x}, \mathbf{z})$ with $\mathbf{z}$ as in Definition 1.19, that is

$$\mathcal{A}_t(\mathbf{x}, \mathbf{y}) = \mathrm{GRS}_{n-t}(\mathbf{x}, \mathbf{z}) \cap (\mathbb{F}_q)^n.$$

The codewords of $\mathscr{C}$ can be folded as codewords of $\mathrm{GRS}_{n-t}(\mathbf{x}, \mathbf{z})$, since $\mathbf{z}$ satisfies for all $0 \leqslant i \leqslant n-1$:

$$\begin{aligned} z_{\sigma(i)} &= \frac{1}{y_{\sigma(i)}} \frac{1}{\prod_{\substack{0 \leqslant j < n \\ j \neq \sigma(i)}} (x_j - x_{\sigma(i)})} = \frac{1}{y_{\sigma(i)}} \frac{1}{\prod_{\substack{0 \leqslant j < n \\ \sigma(j) \neq \sigma(i)}} (x_{\sigma(j)} - x_{\sigma(i)})} \\ &= \frac{1}{\lambda y_i} \frac{1}{\prod_{\substack{0 \leqslant j < n \\ j \neq i}} (\phi(x_j) - \phi(x_i))} \\ &= \frac{1}{\lambda a^{n-1} y_i} \frac{1}{\prod_{\substack{0 \leqslant j < n \\ j \neq i}} (x_j - x_i)} \\ &= \lambda' z_i. \end{aligned}$$

and $\lambda'^d = 1$. The coordinates of the folded code remain in $\mathbb{F}_q$, and we obtain:

$$\overline{\mathcal{A}_t(\mathbf{x}, \mathbf{y})^\sigma} \subset \mathrm{GRS}_r(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}}) \cap (\mathbb{F}_q)^{n/\ell}.$$

with $r, \widetilde{\mathbf{x}}, \widetilde{\mathbf{y}}$ as in Theorem 5.19.

In general, equality does not hold. We give an example of alternant code whose folded code is not the whole alternant code $\mathrm{GRS}_r(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}}) \cap (\mathbb{F}_q)^{n/\ell}$.

Example 5.20. *We pick $q = 3$ and $m = 2$. $\mathbb{F}_{3^2}$ is built as $\mathbb{F}_3[\omega]/(\omega^2 - (\omega + 1))$. We pick the support and multipliers*

$$\begin{aligned} \mathbf{x} &= (1, 2, \omega^7, \omega^3, \omega^5, \omega, \omega^6, \omega^2), \\ \mathbf{y} &= (\omega^5, \omega^5, \omega^5, \omega^5, \omega^7, \omega^7, \omega^7, \omega^7), \end{aligned}$$

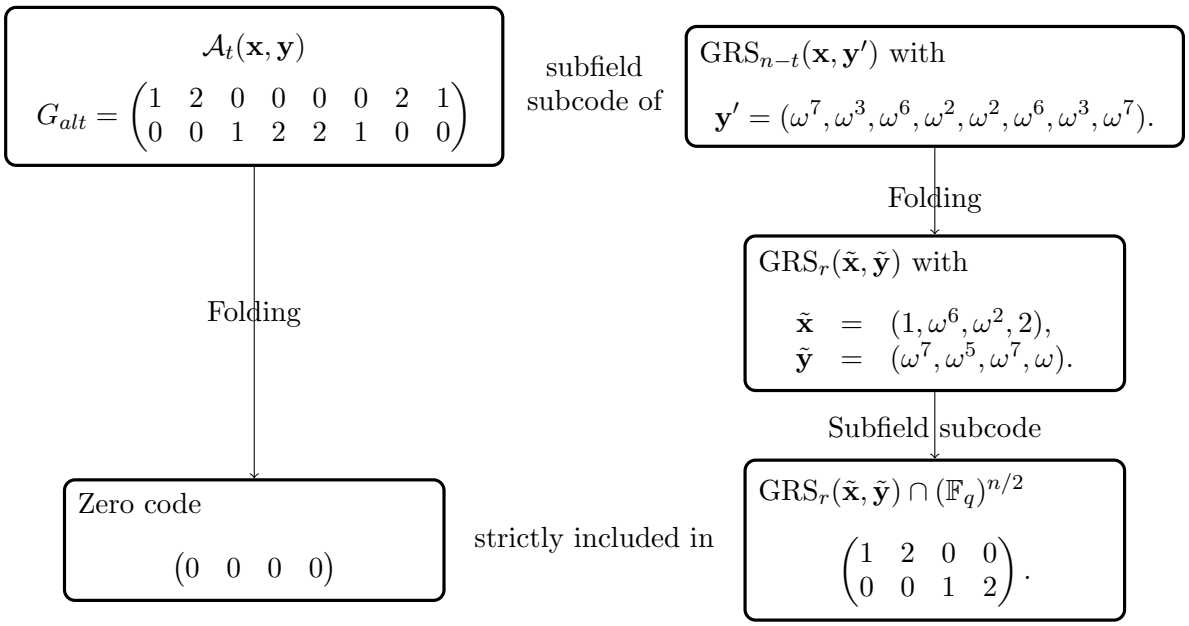


Figure 5.4 – Folded code and dual code of an alternant code.

so that $\mathbf{x}$ *satisfies* $x_{i\oplus_2 1} = \phi(x_i)$ *for* $0 \leqslant i \leqslant 7$ *with* $\phi : z \mapsto -z$ *and* $y_{i\oplus_2 1} = y_i$. *So the codes* $\mathcal{A}_t(\mathbf{x},\mathbf{y}), \mathrm{GRS}_t(\mathbf{x},\mathbf{y})$ *(and its dual* $\mathrm{GRS}_{n-t}(\mathbf{x},\mathbf{y}')$*) admit the permutation* $i \mapsto i \oplus_2 1$ *in their automorphism groups. With* $t = 3$, *we have when the following different folded codes.*

Clearly, the folded code of $\mathcal{A}_3(\mathbf{x},\mathbf{y})$ *is the zero code and it is strictly included in* $\mathrm{GRS}_r(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \cap (\mathbb{F}_q)^{n/2}$.

However, we give in Theorem 5.21 an equality on the folded code of the *dual code* of an alternant code.

Theorem 5.21. *Consider a* QC *or* QM *alternant code* $\mathscr{C}$ *over* $\mathbb{F}_q$ *of length* n. *Let* σ,ℓ,ϕ,a,b,λ,d,$\mathbf{x}$,$\mathbf{y}$ *be as in Theorem 5.19. In such a case, the action of* σ *on* $\{0,1,\ldots,n-1\}$ *has* $\frac{n}{\ell}$ *orbits, each of them being of size* ℓ. *Choose a representative* $i_0, i_1, \ldots, i_{n/\ell-1}$ *in each of these orbits.*

There exists $\mathbf{x}', \mathbf{y}' \in \mathbb{F}_{q^m}^{n/\ell}$ *and a integer* r *such that*

$$\overline{(\mathcal{A}_t(\mathbf{x},\mathbf{y})^{\perp})^{\sigma}} = \Big(\mathcal{A}_r(\mathbf{x}',\mathbf{y}')\Big)^{\perp}$$

with:

— *when* $a = 1$ *then* $r = \left\lfloor \frac{t-\ell}{\ell} \right\rfloor + 1$ *and for all* $j \in \{0,\ldots,n/\ell-1\}$:

$$x'_j = x_{i_j}^{\ell} - b^{\ell-1} x_{i_j} \qquad \text{and} \qquad y'_j = y_{i_j},$$

— *and when* $a \neq 1$ *then* $r = \left\lfloor \frac{t-\ell+d-1}{\ell} \right\rfloor + 1$ *and for all* $j \in \{0,\ldots,n/\ell-1\}$:

$$x'_j = (x_{i_j} - z_0)^{\ell} \qquad \text{and} \qquad y'_j = y_{i_j}(x_{i_j} - z_0)^{\ell-d}.$$

Proof. The proof relies on Lemma 1.21, which gives the following convenient description of the dual of an alternant code:

$$\mathcal{A}_t(\mathbf{x},\mathbf{y})^{\perp} = \mathrm{Tr}_q\Big(\mathrm{GRS}_t(\mathbf{x},\mathbf{y})\Big).$$

The advantage of the trace operator is that is *commutes* with the folding. For any codeword $\mathbf{m}$ of a linear code $\mathscr{C} \subset (\mathbb{F}_{q^m})^n$ with automorphism σ and orbits of size ℓ, the codes $\mathrm{Tr}_q\left(\overline{\mathscr{C}^\sigma}\right)$ and $\overline{\mathrm{Tr}_q\left(\mathscr{C}\right)^\sigma}$ are the set of codewords of coordinates.

$$\sum_{0\leqslant v\leqslant m-1}\sum_{0\leqslant u\leqslant \ell-1} m_{\sigma^u(i_j)}^{q^v} = \sum_{0\leqslant u\leqslant \ell-1}\sum_{0\leqslant v\leqslant m-1} m_{\sigma^u(i_j)}^{q^v}.$$

Therefore, folding $\mathcal{A}_t(\mathbf{x},\mathbf{y})^\perp$ yields

$$\begin{aligned}
\overline{\mathcal{A}_t(\mathbf{x},\mathbf{y})^{\perp\sigma}} &= \overline{\mathrm{Tr}_q\left(\mathrm{GRS}_t(\mathbf{x},\mathbf{y})\right)^\sigma} \\
&= \mathrm{Tr}_q\left(\overline{\left(\mathrm{GRS}_t(\mathbf{x},\mathbf{y})\right)^\sigma}\right) \\
&= \mathrm{Tr}_q\left(\mathrm{GRS}_r(\mathbf{x}',\mathbf{y}')\right) \text{ (Theorem 5.19)} \\
&= \mathcal{A}_r(\mathbf{x}',\mathbf{y}')^\perp \text{ (Lemma 1.21 again)},
\end{aligned}$$

with $r, \mathbf{x}', \mathbf{y}'$ deduced thanks to Theorem 5.19 and as stated Theorem 5.21. □

Finally, and of particular interest for our cryptanalytic purpose, which are detailed in Chapter 6, we prove that the folding preserves also the Goppa structure of a code. More precisely, folding the dual of a Goppa code with an affine-induced automorphism group yields the dual of a Goppa code.

Theorem 5.22. *Consider a Goppa code $\mathscr{C} = \mathscr{G}(\mathbf{x},\Gamma(z))$ of length n associated to the support $\mathbf{x} = (x_i)_{0\leqslant i<n} \in \mathbb{F}_{q^m}^n$ which has an affine induced automorphism group. $\mathscr{C}$ is in particular an alternant code. There exist a support $\mathbf{x} \in \mathbb{F}_{q^m}^n$, multiplier $\mathbf{y} \in \mathbb{F}_{q^m}^n$ and $\lambda \in \mathbb{F}_{q^m}$ an ℓ-th root of unity such that, for all $i \in \{0,1,\ldots,n-1\}$, $x_{\sigma(i)} = \phi(x_i)$, $y_{\sigma(i)} = \lambda y_i$.*

We assume that $a \neq 0$, and $b \neq 0$ if $a = 1$, and that the fixed point z_0 of ϕ does not belong to $\{x_0,\ldots,x_{n-1}\}$. Let ℓ be the order of the permutation on $\mathbf{x}$ induced by ϕ. We have:

1. *ℓ divides n and we set $s = n/\ell$. There are exactly s orbits for the action of σ on the code positions. We denote by $i_0, i_1, \ldots, i_{n/\ell-1}$ a set of representatives for each orbit;*
2. *$\overline{(\mathscr{C}^\perp)^\sigma}$ is the dual of the Goppa code $\mathscr{G}(\mathbf{x}',\gamma(z))$ for $\gamma(z) \in \mathbb{F}_{q^m}[z]$ with:*

$$\begin{aligned}
x'_j &= \begin{cases} x_{i_j}^\ell - b^{\ell-1}x_{i_j} & \text{when } a = 1, \\ (x_{i_j} - z_0)^\ell & \text{otherwise,} \end{cases} \\
\Gamma(z) &= \begin{cases} \gamma(z^\ell - b^{\ell-1}z) & \text{when } a = 1, \\ (z-z_0)^{\ell-d}\gamma\left((z-z_0)^\ell\right) & \text{otherwise} \end{cases}
\end{aligned}$$

where d, in the last case, is the unique integer in $\{0,\ldots,\ell-1\}$ such that $\lambda = a^d$.

Proof. We distinguish between $a = 1$ and $a \neq 1$. In both cases, notice that we can apply Theorem 5.21 to $\mathscr{C}$ which is an alternant code $\mathcal{A}_t(\mathbf{x},\mathbf{y})$ where t is the degree of Γ and $y_i = \frac{1}{\Gamma(x_i)}$. In all cases, folding the dual of $\mathscr{C}$ gives the dual of an alternant code of the form $\mathcal{A}_{t'}(\mathbf{x}',\mathbf{y}')$ for some integer t' and some $\mathbf{x}', \mathbf{y}'$ in $\mathbb{F}_{q^m}^s$. Now we have to prove that there exists a polynomial $\gamma(z) \in \mathbb{F}_{q^m}$ such that $y'_i = \frac{1}{\gamma(x'_i)}$. First, we show that the Goppa polynomial $\Gamma(z)$ satisfies the identity $\Gamma(az+b) = \lambda^{-1}\Gamma(z)$. This is due to the fact that, for all $0 \leqslant i \leqslant n-1$, $y_{\sigma(i)} = \lambda y_i$, that is

$$\Gamma(ax_i + b) = \lambda^{-1}\Gamma(x_i)$$

and $\Gamma(z)$ has degree $t < n$. Therefore, we apply Proposition 5.17. To do so, we separate the cases $a = 1$ and $a \neq 1$.

Case $a = 1$. ℓ is equal to the characteristic p of the field $\mathbb{F}_{q^m}$, α is necessarily equal to 1, $\Gamma(z)$ is of degree a multiple of p and is of the form $\Gamma(z) = \gamma(z^p - b^{p-1}z)$. Notice thanks to Proposition 5.17 that $\mathbf{y}$ satisfies:

$$y_{\sigma(i)} = \frac{1}{\Gamma(x_i + b)} = \frac{1}{\Gamma(x_i)} = y_i.$$

Theorem 5.21 gives that $y'_j = y_{i_j}$ and therefore:

$$y'_j = y_{i_j} = \frac{1}{\Gamma(x_{i_j})} = \frac{1}{\gamma(x_{i_j}^p - b^{p-1}x_{i_j})} = \frac{1}{\gamma(x'_j)}.$$

This implies that $\mathcal{A}_{t'}(\mathbf{x}', \mathbf{y}')$ is nothing but the Goppa code $\mathscr{G}(\mathbf{x}', \gamma(z))$.

Case $a \neq 1$. $\Gamma(z)$ satisfies $\Gamma(az + b) = \alpha\Gamma(z)$ with $\alpha = a^{\ell-d}$, so $\Gamma(z)$ is of the form $\Gamma(z) = (z - z_0)^d \gamma\left((z - z_0)^{\ell-d}\right)$. We use Theorem 5.21 and obtain:

$$y'_j = y_{i_j}(x_{i_j} - z_0)^{\ell-d} = \frac{(x_{i_j} - z_0)^{\ell-d}}{\Gamma(x_{i_j})} = \frac{(x_{i_j} - z_0)^{\ell-d}}{(x_{i_j} - z_0)^{\ell-d}\gamma\left((x_{i_j} - z_0)^{\ell}\right)} = \frac{1}{\gamma(x'_j)}.$$

This implies again that $\mathcal{A}_{t'}(\mathbf{x}', \mathbf{y}')$ is nothing but the Goppa code $\mathscr{G}(\mathbf{x}', \gamma(z))$. $\square$

5.7 Technical Results

We prove the results of Subsection 5.5.1 on affine-invariant polynomials. In the rest of this section, we denote by $I^{\phi,\alpha}_{\leqslant t}[z] \subseteq \mathbb{F}_{q^m}[z]_{\leqslant t}$ be the set of polynomials of degree $\leqslant t$ which satisfy (5.15), i.e. such that $P\big(\phi(z)\big) = \alpha P(z)$. When $\alpha = 1$ we simply write $I^{\phi}_{\leqslant t}[z]$. Finally, when $t < 0$ we adopt the convention that $I_{\leqslant t}[z] = I^{\phi,\alpha}_{\leqslant t}[z] = \{0\}$.

5.7.1 Proof of Proposition 5.17

We first characterize the solutions to Equation (5.15) in the case where $\alpha = 1$.

Lemma 5.23. *Let $\phi(z) = az + b$ be an affine map of finite order ℓ (with $\ell > 1$) defined over a field $\mathbb{F}_{q^m}$. We have*

- *if $a = 1$ and $\mathbb{F}_{q^m}$ is of characteristic ℓ then $I^{\phi}_{\leqslant t}[z] = \big\{R(z^\ell - b^{\ell-1}z) \mid R \in \mathbb{F}_{q^m}[z], \deg R \leqslant t/\ell\big\}$.*
- *if $a \neq 1$ then $I^{\phi}_{\leqslant t}[z] = \big\{R((z - z_0)^\ell) \mid R \in \mathbb{F}_{q^m}[z], \deg R \leqslant t/\ell\big\}$ with z_0 being the unique fixed point of ϕ.*

In other words, the ring of polynomials invariant by an affine map is generated (as a ring) by a single element and the lemma provides this generator explicitly. This result follows from classical results in invariant theory and we derive it from scratch here for completeness. Also, we treat the case where the order ℓ of the group generated by ϕ is divisible by the characteristic of $\mathbb{F}_{q^m}$. This is precisely what happens when $a = 1$, and that is commonly avoided in invariant theory (see for instance [Sha94, Appendix,§4,Prop.1]).

Proof of Lemma 5.23. For $a = 1$, let us first prove $\big\{R(z^\ell - b^{\ell-1}z) \mid \deg R \leqslant t/\ell\big\} \subseteq I^{\phi}_{\leqslant t}[z]$. We consider a polynomial $P(z) = R(z^\ell - b^{\ell-1}z)$ for $R \in \mathbb{F}_{q^m}[z]$ of degree $\leqslant t/\ell$, we have:

$$\begin{aligned} P(z+b) &= R\Big((z+b)^\ell - b^{\ell-1}(z+b)\Big) \\ &= R\Big(z^\ell + b^\ell - b^{\ell-1}z - b^\ell\Big) \\ &= R\Big(z^\ell - b^{\ell-1}z\Big) \\ &= P(z). \end{aligned}$$

We just used the fact that ℓ is the characteristic of $\mathbb{F}_{q^m}$ and therefore $(z+b)^\ell = z^\ell + b^\ell$.

In the case $a \neq 1$, remark that ℓ is also the order of a. The reason is that, for all $u \geqslant 0$, $\phi^u(z) = a^u(z - z_0) + z_0$, where $z_0 \in \mathbb{F}_{q^m}$ is such that $\phi(z_0) = z_0$. Then, we consider a polynomial $P(z) = R\big((z - z_0)^\ell\big)$ for $R \in \mathbb{F}_{q^m}[z]$ of degree $\leqslant t/\ell$ we have:

$$\begin{aligned} P(az+b) &= R\Big((az + b - z_0)^\ell\Big) \\ &= R\Big((az + b - az_0 - b)^\ell\Big) \\ &= R\Big(a^\ell(z - z_0)^\ell\Big) \\ &= R\big((z - z_0)^\ell\big) \\ &= P(z). \end{aligned}$$

Let us prove now the reverse inclusion. Let P be a polynomial which is invariant by ϕ. Consider now a non constant polynomial A of smallest degree which is invariant by ϕ. Such a

polynomial necessarily exists since the set of polynomials which are non constant and which are invariant by ϕ is non empty. In contains $z^\ell - b^{\ell-1}z$ in the case $a = 1$ and $(z - z_0)^\ell$ in the case $a \neq 1$. Perform the division of P by A. We can write

$$P(z) = A(z)P_1(z) + P_2(z) \tag{5.18}$$

with $\deg P_2 < \deg A$. Observe now that

$$P(az + b) = A(az + b)P_1(az + b) + P_2(az + b). \tag{5.19}$$

It holds that $P(az + b) = P(z)$ and $A(az + b) = A(z)$. We deduce by subtracting the second equation to the first one, that we have

$$A(z)\left(P_1(az + b) - P_1(z)\right) = P_2(z) - P_2(az + b).$$

Since the degree of $S(z) = P_2(z) - P_2(az+b)$ is less than the degree of A, this can only happen if $P_1(az + b) - P_1(z) = 0$, *e.g.* P_1 is invariant under ϕ and therefore also P_2. Since A is a non constant polynomial of smallest degree which is invariant under ϕ and since $\deg P_2 < \deg A$, this implies that P_2 is constant. By carrying on this process (i.e. dividing P_1 by A) we eventually obtain that P is a polynomial in A.

We finish the proof by proving that A can be chosen to be $A(z) = z^\ell - b^{\ell-1}z$ in the case $a = 1$ and $A(z) = (z - z_0)^\ell$ otherwise. Let us first prove this for $a = 1$. We can add any constant to A, it is still invariant under ϕ. We may therefore assume that $A(0) = 0$. We can also assume that A is monic. Let us observe now that $0 = A(0) = A(b) = A(2b) = \ldots = A\big((\ell - 1)b\big)$ by the invariance of A under $z \mapsto z + b$. This implies that A is a multiple of $z(z - b)\cdots(z - b(\ell - 1))$. A is therefore of degree greater than or equal to ℓ. The polynomial $z^\ell - b^{\ell-1}z$ is of degree ℓ, is invariant under ϕ and is a multiple of $z(z - b)\ldots(z - b(\ell - 1))$. Therefore $A(z) = z^\ell - b^{\ell-1}z$.

Consider now the case $a \neq 1$. Without loss of generality (by adding a suitable constant as in the case $a = 0$) we may assume that $A(c) = 0$, where c is some element of $\mathbb{F}_{q^m}$ such that the orbit of c under ϕ is of size ℓ. By the invariance of A under ϕ this implies that $0 = A(c) = A(\phi(c)) = \cdots = A\big(\phi^{\ell-1}(c)\big)$. This implies that $A(z)$ is divisible by $(z - c)(z - \phi(c))\cdots\big(z - \phi^{\ell-1}(c)\big)$. Therefore A is of degree ℓ at least. Since $(z - z_0)^\ell$ is of degree ℓ and is invariant by ϕ we can choose $A(z) = (z - z_0)^\ell$. □

This proves Proposition 5.17 when $\alpha = 1$. Let us prove now this proposition in general.

Proof of Proposition 5.17. Denote by ϕ the affine map $z \mapsto az + b$. First of all, let us notice that if there exists some polynomial $P(z)$ satisfying the equation $P(\phi(z)) = \alpha P(z)$ for some α, then necessarily such an α satisfies $\alpha^\ell = 1$. This follows at once from the fact that we have $P(z) = P\left(\phi^\ell(z)\right) = \alpha^\ell P(z)$. This also implies that the order of α divides ℓ. There are now two cases to consider.

Case $a = 1$: then the order ℓ of ϕ is necessarily equal to the characteristic of $\mathbb{F}_{q^m}$ and there is no element, apart from 1, whose order divides ℓ. In this case, Lemma 5.23 implies Proposition 5.17.

Case $a \neq 1$: in such a case the order of a is equal to ℓ and a is a primitive ℓ-th root of unity. Since α is an ℓ-th root of unity, there exists in this case an integer d in the range $[0, \ldots, \ell - 1]$ such that $\alpha = a^d$. Consider now a polynomial which is such that

$$P\left(\phi(z)\right) = \alpha P(z). \tag{5.20}$$

If $\alpha = 1$, then we can use directly Lemma 5.23 and we are done. Otherwise, observe that from the fact that $\phi(z_0) = z_0$ we deduce that

$$P(z_0) = P(\phi(z_0)) = \alpha P(z_0).$$

This implies that $P(z_0) = 0$. Define now a polynomial P_1 by $P(z) = (z - z_0)P_1(z)$. Observe now that on the one hand

$$P(az + b) = (az + b - z_0)P_1(az + b) = a(z - z_0)P_1(az + b)$$

and that on the other hand

$$P(az + b) = \alpha P(z) = a^d(z - z_0)P_1(z).$$

Putting both equations together, we obtain

$$P_1(az + b) = a^{d-1}P_1(z)$$

If $d \neq 1$ we can carry on this process on P_1, deduce from the previous equation that $P_1(z_0) = 0$ and deduce by induction on d that $P(z)$ has a zero of order at least d at z_0 and that the polynomial $P_d(z)$ defined by $P_d(z) = \frac{P(z)}{(z-z_0)^d}$ satisfies the equation

$$P_d(az + b) = P_d(z).$$

We apply Lemma 5.23 to P_d and derive from it that P should be of the form

$$P(z) = (z - z_0)^d Q\left((z - z_0)^\ell\right),$$

where Q is any polynomial of degree $\frac{\deg P - d}{\ell}$. Conversely, any polynomial P of this form is readily seen to verify (5.20). □

5.7.2 Proof of Proposition 5.18

To prove Proposition 5.18, we need the following lemma.

Lemma 5.24. $1^k + 2^k + \cdots + (p-1)^k \equiv 0 \pmod p$ *for every integer k which is not a multiple of $p - 1$ whereas* $1^k + 2^k + \cdots + (p - 1)^k \equiv -1 \pmod p$ *otherwise.*

Proof. Recall that the multiplicative group $\mathbb{F}_p^\times$ is generated by a single element α which is of order $p - 1$. The mapping

$$\begin{array}{rcl} \phi_k : \mathbb{F}_p^\times & \to & \mathbb{F}_p^\times \\ x & \mapsto & x^k \end{array}$$

maps therefore $\mathbb{F}_p^\times$ to a subgroup of $\mathbb{F}_p^\times$ different from the trivial subgroup consisting only of 1 if and only if k is not a multiple of $p-1$. In other words, if k is a multiple of $p-1$, we have $s^k \equiv 1 \pmod p$ for any $s \in \{1, \ldots, p-1\}$. This implies that $1^k+2^k+\cdots+(p-1)^k \equiv p-1 \equiv -1 \pmod p$. Assume now that k is not a multiple of $p - 1$. Thus $\phi_k(\mathbb{F}_p^\times)$ is a subgroup of $\mathbb{F}_p^\times$ of size a divisor $\ell > 1$ of $p - 1$. Since $\mathbb{F}_p^\times$ is generated by α, $\phi_k(\mathbb{F}_p^\times)$ is generated by $\beta = \alpha^k$ and we have

$$\begin{aligned} 1^k + 2^k + \cdots + (p - 1)^k &\equiv \frac{p-1}{\ell}\left(1 + \beta + \cdots + \beta^{\ell-1}\right) \pmod p \\ &\equiv \frac{(p-1)(\beta^\ell - 1)}{\ell(\beta - 1)} \pmod p \\ &\equiv 0 \pmod p \end{aligned}$$

□

Let us prove now Proposition 5.18 when $a = 1$.

Proof. Let us first compute $S(z^t)$, where t is some nonnegative integer.

$$\begin{aligned} S(z^t) &= \sum_{s=0}^{p-1}(z+sb)^t = z^t + \sum_{s=1}^{p-1}\sum_{i=0}^{t}\binom{t}{i}z^{t-i}(sb)^i = \sum_{s=1}^{p-1}\sum_{i=1}^{t}\binom{t}{i}z^{t-i}(sb)^i \\ &= \sum_{i=1}^{t} b^i\binom{t}{i}\left(\sum_{s=1}^{p-1}s^i\right)z^{t-i} = \sum_{i=p-1}^{t} b^i\binom{t}{i}\left(\sum_{s=1}^{p-1}s^i\right)z^{t-i} \end{aligned} \tag{5.21}$$

where the last equation follows by using Lemma 5.24 which allows us to write $\sum_{s=1}^{p-1} s^i = 0$ when i is in the range $[1..p-2]$ and when the sum is performed over a field of characteristic p. This implies immediately that $S(\mathbb{F}_{q^m}[z]_{\leqslant t}) \subseteq \mathbb{F}_{q^m}[z]_{\leqslant t-p+1}$. Since $S(Q(z))$ is obviously invariant by ϕ for any polynomial $Q(z) \in \mathbb{F}_{q^m}[z]$, we know from Lemma 5.23 that it is of the form $S(Q(z)) = R(z^p - b^{p-1}z)$ for some polynomial R in $\mathbb{F}_{q^m}[z]$. Its degree is therefore a multiple of p. This implies that we actually obtain the refined inclusion

$$S(\mathbb{F}_{q^m}[z]_{\leqslant t}) \subseteq I_{\leqslant \left\lfloor \frac{t-p+1}{p} \right\rfloor p}[z]. \tag{5.22}$$

Equality is proven by dimension considerations. It follows from Lemma 5.23 that $I_{\leqslant t}[z]$ is a vector space which is of dimension $\lfloor t/p \rfloor + 1$. The calculation (5.21) performed above also shows that $S(z^{(k+1)p-1})$ is a polynomial of degree kp (since the coefficient of z^{kp} which is equal to $b^{p-1}\binom{(k+1)p-1}{p-1}\sum_{s=1}^{p-1} s^{p-1}$ by (5.21) can be shown to be different from 0 by using the fact proven in Lemma 5.24 which says that $1^{p-1} + 2^{p-1} + \cdots + (p-1)^{p-1} \equiv -1 \pmod p$). This can be used to obtain that

$$\dim S(\mathbb{F}_{q^m}[z]_{\leqslant t}) \geqslant \left\lfloor \frac{t-p+1}{p} \right\rfloor + 1 = \dim I_{\leqslant \left\lfloor \frac{t-p+1}{p} \right\rfloor p}[z].$$

This together with (5.22) implies that

$$S(\mathbb{F}_{q^m}[z]_{\leqslant t}) = I_{\leqslant \left\lfloor \frac{t-p+1}{p} \right\rfloor p}[z],$$

which concludes the proof. □

Now, we deal with the case $a \neq 1$. Let us calculate

$$\begin{aligned} S(z^t) &= \sum_{i=0}^{\ell-1} a^{di}\left(a^i(z-z_0)\right)^t, \\ &= (z-z_0)^t \sum_{i=0}^{\ell-1} a^{(d+t)i}. \end{aligned}$$

This sum is equal to 0 as long as $d + t \not\equiv 0 \pmod \ell$ and is equal to $(\ell \mod p)\,(z-z_0)^t$ when $d + t \equiv 0 \pmod \ell$. The polynomial $S\big(P(z)\big)$ is therefore a polynomial of degree $\ell - d + \left\lfloor \frac{\deg P - \ell + d}{\ell} \right\rfloor \ell$ of the form

$$S(P(z)) = (z-z_0)^{\ell-d} \sum_{i=0}^{\left\lfloor \frac{\deg P-\ell+d}{\ell} \right\rfloor} a_i(z-z_0)^{i\ell} \tag{5.23}$$

when $\deg P \geqslant \ell - d$ and is equal to zero otherwise. We conclude the proof by noting that the term $\sum_{i=0}^{\left\lfloor \frac{\deg P-\ell+d}{\ell} \right\rfloor} a_i(z-z_0)^{i\ell}$ is a polynomial which is invariant by ϕ by Lemma 5.23.

Chapter 6

Algebraic cryptanalysis of McEliece schemes using QM Alternant codes

This chapter presents results from an article written with Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret and Jean-Pierre Tillich : *Structural Cryptanalysis of McEliece Schemes with Compact Keys*, accepted for publication at *Designs, Codes and Cryptography*, ([FOP+15]).

We present an algebraic cryptanalysis of code-based encryption and signature schemes proposed recently in [MB09, BCMN10, BLM11]. The public codes used in those schemes all belong to the class of alternant codes with symmetries described in Chapter 5. Compared to standard alternant codes, those codes admit compact representations, so that the associated public-key scheme has a reduced key size. With the notations of Definition 5.3, we say that [MB09, BCMN10] quasi-dyadic (QD) codes, and [BLM11] quasi-monoidic (QM) codes.

The central idea of our cryptanalysis is to use iteratively the folding construction of Chapter 5. Thanks to Theorem 5.21, we know that the folding reduces the code parameters but preserves the secret elements, so that we can recover the secret key on the smallest possible code obtained by successive foldings. For instance, we can reduce the key-recovery of a QD Goppa code of length 8192 and dimension 4096 (parameters suggested in [MB09]) to the key-recovery on a QD Goppa code of length 64 and dimension 32. This is formalized in Theorem 6.1 (Section 6.1).

In Section 6.2, we exploit this result to mount an attack on QD and QM codes. The strategy is to perform the key-recovery on the folded code. We use for this the algebraic attacks of Chapter 4, which profits a lot from the folding method, since it reduces the amount of unknowns to introduce. In [FOPT10a], the idea is to write the system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ (as described in Chapter 4) and simplify it (with Equations (5.10) and (5.11) of Theorem 5.10). With our method, modelling the folded code requires fewer variables than the simplified system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$. For example, pick a code defined over $\mathbb{F}_2$ with $m = 13$, $n = n_0 t$ with $n_0 = 511$ and $t = 2^4$ (these are good signature parameters). The simplified $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ of [FOPT10a] contains 525 variables, the system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ obtained from the folded code contains 521 variables. This is better, but we can do much better thanks to Chapter 4, where we described refined algebraic modellings $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}, \mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}, \mathsf{McE}_{\mathbf{X},\mathbf{Y}'} \ldots$ according to the structure of the code (Table 4.1). With those parameters, our system $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ obtained from the folded code contains only 25 variables (the system $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ describing the original code would contain 29). In Figure 6.1, we summarize both methods. In practice, our method allows to break far more parameters. Details on our experimental results can be found in Paragraph 6.2.3.

As final remark, we point out that the sizes of the systems that we obtain with our method,

$$\mathbf{G} = \left(\begin{array}{c|c|c} \cdots & \mathbf{M}_i & \cdots \\ \hline & \vdots & \end{array}\right) \xrightarrow[\text{of [FOPT10a]}]{\text{Algebraic modelling}}$$

	Simplified $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$
$\mathbf{X}$	$n/t + \lambda$
$\mathbf{Y}$	$(n-k)/t$

$\mathbf{G}$ a $k \times n$ matrix,
$\mathbf{M}_i$ a $t \times t$ matrix with $t = p^\lambda$

Folding
(Theorem 6.1)

$$\left(\begin{array}{ccc} \cdots & m_i & \cdots \\ & \vdots & \end{array}\right) \xrightarrow[\text{of Chapter 4}]{\text{Algebraic modelling}}$$

	$\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$	$\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$
$\mathbf{X}$	n/t	$(n-k)/t$
$\mathbf{Y}$	$(n-k)/t$	$(n-k)/t$

$k/t \times n/t$ matrix

Figure 6.1 – Comparison of the algebraic systems for a key-recovery of a $[n, k, t]_q$ QC/QM code.

and the complexity of our attack, depend *only* on the reduced parameters $n/t, k/t$ of the code (where $t = p^\lambda$ is the block size in the public matrix). Consequently, the key-security is reduced to that of a small alternant code with same key size and *without* symmetries. Therefore, the folding construction questions the very use of QD and QM alternant codes for cryptography.

6.1 Iterated Folded Code

Building a generator matrix of the folded code simply consists in summing the coordinates of each row of a generator matrix of $\mathscr{C}$ over the orbits of σ. This can be done efficiently. We point out that Theorem 5.21 applies on duals of alternant codes. In practice, folding the dual of a code simply means that we fold a parity-check matrix of the code. We now apply Theorems 5.21 and 6.1 several times to prove that folding successively the dual of QM alternant or Goppa codes according to all the generators $\sigma_1, \sigma_p \ldots, \sigma_{p^{\lambda-1}}$ of the permutation group introduced in Theorem 5.10 yields the dual of a QM Goppa codes with smaller parameters. The final reduction factor is equal to the size of the automorphism group of the initial code.

Theorem 6.1. *Let $n = n_0 p^\lambda, \mathbf{x} = (x_i)_{0 \leqslant i < n}$, $\alpha_0, \ldots, \alpha_{\lambda-1} \in \mathbb{F}_{q^m}$, and $\Gamma(z)$ of degree $t_0 p^\lambda$ be as in Theorem 5.10. The Goppa code $\mathscr{G}(\mathbf{x}, \Gamma(z))$ has a quasi-monoidic parity-check matrix $\mathbf{H} = (h_{ij})_{\substack{0 \leqslant i < t_0 p^\lambda \\ 0 \leqslant j < n}}$ with monoidic blocks of size p^λ. For $u \in \{0, \ldots, \lambda\}$, let $\mathbf{H}^u = (h^u_{ij})_{\substack{0 \leqslant i < t_0 p^{\lambda-u} \\ 0 \leqslant j < n_0 p^{\lambda-u}}}$ be the matrix defined by $h^u_{i,j} = \sum_{\ell=0}^{p^u-1} h_{ip^u, jp^u+\ell}$. It holds that $\mathbf{H}^u$ is a parity-check matrix of a Goppa code $\mathscr{G}(\mathbf{x}^u, \Gamma_u(z))$ for some $\mathbf{x}^u \in \mathbb{F}_{q^m}^{n_0 p^{\lambda-u}}$, and $\Gamma_u(z)$ a polynomial of degree $t_0 p^{\lambda-u}$. Also, $\mathbf{x}^u$ and $\Gamma_u(z)$ satisfy the hypotheses of Theorem 5.10: $\mathbf{x}^u$ satifies Equations (5.10) with $\alpha^u_0, \ldots, \alpha^u_{\lambda-u-1} \in \mathbb{F}_{q^m}$, and $\Gamma_u(z)$ satisfies (5.12).*

More explicitly, they are defined inductively by $\mathbf{x}^0 = \mathbf{x}$, $\alpha^0_j = \alpha_j$ for $0 \leqslant j \leqslant \lambda - 1$, $\Gamma_0(z) = \Gamma(z)$, and

$$\begin{aligned}
\alpha^{u+1}_j &= \left(\alpha^u_j\right)^p - (\alpha^u_0)^{p-1}\, \alpha^u_j, \text{ for } 0 \leqslant j \leqslant \lambda - u - 1, \\
x^{u+1}_i &= \left(x^u_{ip}\right)^p - (\alpha^u_0)^{p-1}\, x^u_{ip} \text{ for } 0 \leqslant i \leqslant t_0 p^{\lambda-u-1} - 1, \\
\Gamma_u(z) &= \Gamma_{u+1}\left(z^p - (\alpha^u_0)^{p-1}\, z\right).
\end{aligned}$$

So, if we view these Goppa codes as alternant codes, that is $\mathscr{G}(\mathbf{x}, \Gamma(z)) = \mathcal{A}_{t_0 p^\lambda}(\mathbf{x}, \mathbf{y})$ and $\mathscr{G}(\mathbf{x}^u, \Gamma_u(z)) = \mathcal{A}_{t_0 p^{\lambda-u}}(\mathbf{x}^u, \mathbf{y}^u)$, then the multipliers $\mathbf{y}^u$ are given by

$$\mathbf{y}^u_i = \mathbf{y}_{p^u i} \text{ for } 0 \leqslant i \leqslant t_0 p^{\lambda-u} - 1.$$

and $\mathbf{y}_i$ is equal to $\mathbf{y}_{\lfloor i/p^\lambda \rfloor p^\lambda}$ for all $i \in \{0, \ldots, n-1\}$.

Proof. Let us prove by induction on u the following statements:
(i) the code with parity-check matrix $\mathbf{H}^u$ is a Goppa code $\mathscr{G}(\mathbf{x}^u, \Gamma_u(z))$ where $\Gamma_u(z)$ is a polynomial which satisfies (5.12):

$$\Gamma_u(z) = \Gamma_u(z + \alpha^u_0) = \cdots = \Gamma_u(z + \alpha^u_{\lambda-u-1}) \tag{6.1}$$

for some $\alpha^u_0, \ldots, \alpha^u_{\lambda-u-1}$ in $\mathbb{F}_{q^m}$ which are independent over $\mathbb{F}_p$,
(ii) $\mathbf{x}^u$ is obtained thanks to (5.10)

$$x^u_i = x^u_{\lfloor i/p^{\lambda-u} \rfloor p^{\lambda-u}} + \sum_{j=0}^{\lambda-1-u} i_j \alpha^u_j \text{ for } 0 \leqslant i \leqslant t_0 p^{\lambda-u} - 1, \tag{6.2}$$

with $(i_0, \ldots, i_{\lambda-u-1}) \in \mathbb{F}_p^\lambda$ being the representation of i mod $p^{\lambda-u}$ in base p, i.e. $i \equiv \sum_{j=0}^{\lambda-u-1} i_j p^j$ mod $p^{\lambda-u}$.

For $u = 0$ this holds by assumption on $\mathbf{H}$ and by using the way $\mathbf{x}$ was constructed, that is from independent (over $\mathbb{F}_p$) elements $\alpha_0, \dots \alpha_{\lambda-1}$ in $\mathbb{F}_p$ for which we have

$$x_i = x_{\lfloor i/p^\lambda \rfloor p^\lambda} + \sum_{j=0}^{\lambda-1} i_j \alpha_j,$$

where $(i_0, \dots, i_{\lambda-1}) \in \mathbb{F}_p^\lambda$ is the representation of i mod p^λ in base p.

Suppose now that u is an integer in the range $[0..\lambda]$ and that $\mathbf{H}^u$ is a parity-check matrix of a Goppa code $\mathscr{G}(\mathbf{x}^u, \Gamma_u(z))$ for some $\mathbf{x}^u \in \mathbb{F}_{q^m}^{n_0 p^{\lambda-u}}$ where there exist $\alpha_0^u, \dots, \alpha_{\lambda-u-1}^u$ in $\mathbb{F}_{q^m}$ which are independent over $\mathbb{F}_p$ and for which (6.1) and (6.2) hold.

It can also be viewed as the alternant code $\mathcal{A}_{t_0 p^{\lambda-u}}(\mathbf{x}^u, \mathbf{y}^u)$ with $y_i^u = \frac{1}{\Gamma_u(x_i^u)}$. Thanks to (6.2) of part (ii) of the induction hypothesis, we know by Theorem 5.21 that $\mathbf{H}^{u+1}$ is a parity-check matrix of the alternant code $\mathcal{A}_{t_0 p^{\lambda-u-1}}(\mathbf{x}^{u+1}, \mathbf{y}^{u+1})$ with

$$x_i^{u+1} = \left(x_{ip}^u\right)^p - (\alpha_0^u)^{p-1} x_{ip}^u, \tag{6.3}$$

$$y_i^{u+1} = y_{ip}^u. \tag{6.4}$$

Notice that the induction assumption and (6.3) imply that

$$\begin{aligned}
x_i^{u+1} &= \left(x_{ip}^u\right)^p - (\alpha_0^u)^{p-1} x_{ip}^u \\
&= \left(x_{\lfloor i/p^{\lambda-u-1} \rfloor p^{\lambda-u}}^u + \sum_{j=1}^{\lambda-1-u} i_j \alpha_j^u\right)^p - (\alpha_0^u)^{p-1} \left(x_{\lfloor i/p^{\lambda-u-1} \rfloor p^{\lambda-u}}^u + \sum_{j=1}^{\lambda-1-u} i_j \alpha_j^u\right) \\
&= \left(x_{\lfloor i/p^{\lambda-u-1} \rfloor p^{\lambda-u}}^u\right)^p - (\alpha_0^u)^{p-1} x_{\lfloor i/p^{\lambda-u-1} \rfloor p^{\lambda-u}}^u - \sum_{j=1}^{\lambda-1-u} i_j \left(\left(\alpha_j^u\right)^p - (\alpha_0^u)^{p-1} \alpha_j^u\right) \\
&= x_{\lfloor i/p^{\lambda-u-1} \rfloor p^{\lambda-u-1}}^{u+1} - \sum_{j=0}^{\lambda-2-u} i'_j \alpha_j^{u+1}
\end{aligned}$$

where $(i'_0, \dots, i'_{\lambda-u-2}) \in \mathbb{F}_p^{\lambda-u-1}$ is the representation of i mod $p^{\lambda-u-1}$ in base p and $\alpha_j^{u+1} = \left(\alpha_j^u\right)^p - (\alpha_0^u)^{p-1} \alpha_j^u$. By (6.4), we deduce that

$$y_i^{u+1} = \frac{1}{\Gamma_u(x_{ip}^u)} \tag{6.5}$$

Since the polynomial $\Gamma_u(z)$ is invariant by the shift $x \mapsto x + \alpha_0^u$ we can write $\Gamma_u(z)$ in the form $\Gamma_u(z) = \Gamma_{u+1}\left(z^p - (\alpha_0^u)^{p-1} z\right)$ for some polynomial Γ_{u+1} of degree $t_0 p^{\lambda-u-1}$ (Proposition 5.17). This implies that

$$y_i^{u+1} = \frac{1}{\Gamma_{u+1}\left(\left(x_{ip}^u\right)^p - (\alpha_0^u)^{p-1} x_{ip}^u\right)} = \frac{1}{\Gamma_{u+1}(x_i^{u+1})}$$

and therefore $\mathcal{A}_{t_0 p^{\lambda-u-1}}(\mathbf{x}^{u+1}, \mathbf{y}^{u+1})$ is also the Goppa code $\mathscr{G}(\mathbf{x}^{u+1}, \Gamma_{u+1}(z))$ □

We show on a very small example how the succesive steps of Theorem 6.1 work.

Example 6.2 (Iterated Folded code). *We fold the quasi-monoidic* $\mathscr{C}_2 = \mathscr{A}_{t_2}(\widetilde{\mathbf{x}}^{(2)}, \widetilde{\mathbf{y}}^{(2)}) = \mathscr{G}(\widetilde{\mathbf{x}}^{(2)}, \Gamma_2(z))$ *of Example 5.12 in Chapter 5.*$\mathscr{C}_2$ *admits as generator matrix* $G^{(2)}$*:*

$$G^{(2)} = \left(\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc|cccc}
1&0&0&0&0&0&0&0&1&0&1&1&0&0&0&1&0&1&1&1&0&0&0&1&1&1&1&1\\
0&1&0&0&0&0&0&0&0&1&1&1&0&0&1&0&1&0&1&1&0&0&1&0&1&1&1&1\\
0&0&1&0&0&0&0&0&1&1&1&0&0&1&0&0&1&1&0&1&0&1&0&0&1&1&1&1\\
0&0&0&1&0&0&0&0&1&1&0&1&1&0&0&0&1&1&1&0&1&0&0&0&1&1&1&1\\
\hline
0&0&0&0&1&0&0&0&0&1&1&0&1&1&1&0&0&1&0&0&0&1&1&1&0&1&0&1\\
0&0&0&0&0&1&0&0&1&0&0&1&1&1&0&1&1&0&0&0&1&0&1&1&1&0&1&0\\
0&0&0&0&0&0&1&0&1&0&0&1&1&0&1&1&0&0&0&1&1&1&0&1&0&1&0&1\\
0&0&0&0&0&0&0&1&0&1&1&0&0&1&1&1&0&0&1&0&1&1&1&0&1&0&1&0
\end{array}\right)$$

To construct the folded matrix, one only needs to sum up the coefficients of the generator matrix $\mathbf{G}^{(2)}$ *corresponding to each orbit of* $i \mapsto i \ominus_2 1$ *(that is, the subsets of the code positions of the form* $\{2i, 2i+1\}$*) :*

$$\mathbf{G}^{(1)}_{fold} = \left(\begin{array}{cc|cc|cc|cc|cc|cc|cc}
1&0&0&0&1&0&0&1&1&0&0&1&0&0\\
0&1&0&0&0&1&1&0&0&1&1&0&0&0\\
\hline
0&0&1&0&1&1&0&1&1&0&1&0&1&1\\
0&0&0&1&1&1&1&0&0&1&0&1&1&1
\end{array}\right).$$

Thanks to Theorems 6.1, we known that $G^{(1)}_{fold}$ *generates an alternant code defined by the following vectors:*

$$\begin{array}{rcl}
\mathbf{x}^{(1)} &=& (0, \omega^{30}, \omega^{28}, \omega^2, \omega, \omega^4, \omega^{26}, \omega^5, \omega^{19}, \omega^7, \omega^{18}, \omega^{10}, \omega^{17}, 1),\\
\mathbf{y}^{(1)} &=& (\omega^{29}, \omega^{29}, \omega^3, \omega^3, 1, 1, \omega^{26}, \omega^{26}, \omega^{14}, \omega^{14}, \omega, \omega, \omega^5, \omega^5).
\end{array}$$

and with Goppa polynomial $\Gamma_1(z) = z^2 + \omega^{30} z + \omega^2$*.*

In our example, the order of symmetry introduced in $\mathscr{C}_2$ *was* 2^2*, as* $\widetilde{\mathbf{x}}^{(2)}$ *satisfies both relations* $\widetilde{x}^{(2)}_{i\ominus_2 1} = \widetilde{x}^{(2)}_i + \alpha_0$ *and* $\widetilde{x}^{(2)}_{i\ominus_2 2} = \widetilde{x}^{(2)}_i + \alpha_1$*. In this case, it is not hard to see that* $\mathbf{x}^{(1)}$ *satisfies* $x^{(1)}_{i\ominus 1} = x^{(1)}_i + \alpha_1^2 - \alpha_0\alpha_1$*. This shows that the code spanned by* $G^{(1)}_{fold}$ *can still be folded. This double-folded code can be obtained either by summing the coefficients of* $G^{(1)}_{fold}$ *over the orbits* $\{2i, 2i+1\}$*, or directly on the matrix* $G^{(2)}$*, by considering the orbits* $\{4i, 4i+1, 4i+2, 4i+3\}$*:*

$$\mathbf{G}^{(0)}_{fold} = \left(\begin{array}{ccccccc}
1&0&1&1&1&1&0\\
0&1&0&1&1&1&0
\end{array}\right).$$

Then, thanks to Theorems 6.1, we know that $G^{(0)}_{fold}$ *is an alternant code, with private elements:*

$$\mathbf{x}^{(0)} = (0, \omega^{30}, \omega^5, 1, \omega^{26}, \omega^{28}, \omega^{17}),$$
$$\mathbf{y}^{(0)} = (\omega^{29}, \omega^3, 1, \omega^{26}, \omega^{14}, \omega, \omega^5).$$

and $\Gamma_0(z) = z + \omega^2$*.*

When considering a cryptographic scheme with public key the $\mathscr{C}_2$ of the previous example, it is clear that a public user has easy access to the matrices $\mathbf{G}^{(1)}_{fold}$ and $\mathbf{G}^{(0)}_{fold}$, but the private vectors $\mathbf{x}^{(1)}, \mathbf{y}^{(1)}, \mathbf{x}^{(0)}, \mathbf{y}^{(0)}$, along with the Goppa polynomials, remain unkown. However, any attack which permits to find $\mathbf{y}^{(0)}$ (or $\mathbf{y}^{(1)}$) gives immediate access to $\mathbf{y}$ thanks to Relation (6.4), and then $\mathbf{x}$ by solving a linear system (Fact 4.1).

6.2 Application to the Cryptanalysis of QM alternant codes

We use Theorem 6.1 to recover the secret key of QM alternant codes. We explained that it is sufficient to recover the secret key of a folded code of the original code. To perform this key-recovery in practice, we discard the SSA attack (Paragraph 2.3.2) for the codes that we target. Indeed, as those are folded codes of longer codes, folded codes are always non full-support codes, for which the complexity analysis in Section 4.4 of Chapter 4 revealed that algebraic attacks are more efficient.

6.2.1 Full Cryptanalysis Strategy

We suppose that the public code was built with Theorem 5.10, so that $t = p^\lambda$ for $\lambda \geqslant 1$. The first part of the attack is to successively fold the public code. We obtain codes with same structure (alternant or Goppa) and same multipliers (Theorem 6.1). After s iterations ($0 \leqslant s \leqslant \lambda - 1$) of the folding process, the resulting code is of order $t_s = t/p^{(\lambda-s)}$, length $n_s = n/p^{(\lambda-s)}$ and dimension $k_s = k/p^{(\lambda-s)}$. Then we choose in Table 4.1 a system corresponding to the structure of the code. One can see that, compared to the original algebraic modelling, the algebraic modelling of the folded code will have fewer variables and equations, same over-determination ratio, and equations of much smaller degrees containing much fewer monomials. For Gröbner bases algorithms [Fau99, Fau02], this is a very good deal (see Paragraph 3.3 of Chapter 3). For instance, compared to FOPT's experimental results [FOPT10a, FOPT10b], we obtained with this new strategy (Table 6.2) a speed-up wich can be as big as 45000 using on-the-shelf computer algebra system Magma [BCP97b], while the results of [FOPT10a, FOPT10b] were obtained with the Fgb software[Fau10]; an optimized C implementation of the F_5 algorithm [Fau02].

Once private support and multipliers are known for the folded code, we also know the multipliers of the public code (Theorem 6.1). Then, Fact 4.1 ensures that one can efficiently recover an equivalent decoder for the public code. We sum-up the strategy as follows (Steps 0) and 6) are exposed in 6.2.2).

Choice of the final $\widetilde{t}$ and of the algebraic system. Regardless of the key recovery strategy, it is natural to consider alternant folded code of degree $\widetilde{t}$ as small as possible. However, there are lower bounds on $\widetilde{t}$. According to Table 4.1, $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ is empty (*i.e* contains no equation) for an alternant code of degree $\widetilde{t} = 1$, and $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ is empty for a code of degree $\widetilde{t} = p$ (i.e. $\lambda = 1$). Thus, we need to take $\widetilde{t}$ not smaller than p for $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ or $\widetilde{t}$ not smaller than p^2 for $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$. Thus, the final $\widetilde{t}$ is determined by the choice to perform or not the structural elimination. We reported for each of our experiments the chosen $\widetilde{t}$. This is the major choice that an attacker has to make. It is actually not always straightforward to predict a priori which system between $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ and $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ is easier to solve. Equations in $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ are of higher degrees than in $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ but contain fewer variables. The relevant parameters to take into account are the characteristic (as the structural elimination increases the bi-degree by a factor of p), and the length of the code (the amount of variables in $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ decreases when the length grows). For various parameters (notably for encryption parameters), we tried both and selected the most efficient.

6.2.2 Extension to all signature parameters

Most of the results presented until now assume that $t = p^\lambda$. Whilst this a very natural requirement, it turns that some parameters proposed for signature in [BCMN10, BLM11] use

Algorithm 7 Cryptanalysis strategy of QM codes

INPUT : A generator matrix of the public code $\mathscr{C}$.
OUTPUT : A polynomial time decoder for the public code (that is $\mathbf{x}'$, $\Gamma'(z)$ such that $\mathscr{C} = \mathscr{G}(\mathbf{x}', \Gamma')$)

0) (If t is not a pure power of p, expand the public QM code into a QM code whose order is a pure power of t. This expanded code has the same support as the public code (Theorem 6.3). We use then this expanded code as a public-key in the next steps.)
1) Fold iteratively the matrix of the public code to obtain a code with the same structure (alternant, or Goppa) but with smaller order of symmetry $\widetilde{t} < t$ (Theorem 6.1).
2) Construct the algebraic system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}, \mathsf{G}_{\mathbf{X},\mathbf{Y}'}$, or $\mathsf{McE}_{\mathbf{X},\mathbf{Y}'}$ (Chapter 4) from the folded code with or without structural elimination (Section 4.2).
3) Use Gröbner bases to recover the multiplier vector of the folded code (results in 6.2.3).
4) Expand the multiplier of the folded code to the multiplier of the public code (Theorem 6.1).
5) Solve the linear system (Fact 4.1) to find a support $\mathbf{x}'$ which allows to construct a decoder for the public code.
6) (If t is not a pure power of p, generate the system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ from the original (i.e. non expanded) public-key matrix, and solve the linear system obtained by plugging the support found at the previous step in $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$. This yields the multipliers of the initial public code, and then construct a decoder for the public-key).

"degenerated" QM codes, where t is not a pure power of p. However, a parity-check matrix of the public code of the form $\mathbf{H} = \left[\frac{1}{g_i - x_j}\right]_{\substack{0\leqslant i\leqslant t-1\\ 0\leqslant j\leqslant n-1}} = (h_{i,j})$ still satisfies $h_{i,j} = h_{0,j\ominus i}$ for $0 \leqslant i \leqslant t-1$ and $0 \leqslant j \leqslant n-1$. Then, it turns out that the automorphism group is not of cardinality t but $\gcd(t, p^\lambda)$ where λ is such that $p^{\lambda-1} < t < p^\lambda$. We recall that a central tool in the construction of QM codes is the sequence of $\mathbb{F}_p$-independent elements $\alpha_0, \ldots, \alpha_{\lambda-1} \in \mathbb{F}_{q^m}$ (Theorem 6.1). They generate a group G of size p^λ whose elements are defined for $\ell \in [0, \ldots, p^\lambda - 1]$ by $g_\ell = \sum_{j=0}^{\lambda-1} \ell_j \alpha_j$, when $\ell = \sum_{j=0}^{\lambda-1} \ell_j p^j$ is the decomposition in base p of ℓ. The support $\mathbf{x}$ is chosen so as to satisfy Equation (5.10). Then, the Goppa polynomial is derived from its roots A that are picked in G. The case $t = p^\lambda$ corresponds to a Goppa polynomial $\Gamma(z)$ whose roots A are all the elements of G. The fact that its roots form a group is the crucial point for proving Equation (5.11), and to exhibit the structure of the automorphism group. The cases where $t \neq p^\lambda$ are those when A is a sub-set of G (and not a sub-group). The associated Goppa code $\mathscr{G}(\mathbf{x}, \prod_{0\leqslant \ell\leqslant t-1}(z - g_\ell))$ is not stable by the expected permutations σ_ℓ, so folding the code is not possible any more. So, the idea is to select a subspace of the codewords that is stable by more permutations.

As a consequence, we cannot fold the public code as many times as previously. To overcome this technical problem, we can however exploit the specific features of the QM codes used in [BCMN10, BLM11]. We can retrieve an alternant code with the same support as the public code, but with a Goppa polynomial of degree p^λ, where we define $\lambda > 0$ as the smallest integer such that $t < p^\lambda$. We shall call $\mathscr{C}_{pub}^{(t)}$ the public code, and $\mathscr{C}^{(p^\lambda)}$ the code that we are trying to recover.

Theorem 6.3. *Let* $\mathbf{x}$ *and* $G = \{g_0, \ldots, g_{p^\lambda-1}\}$ *be as previously. Let* $\mathbf{H} = (h_{i,j})$ *be any*

$(n-k)\times n$ parity-check matrix of $\mathscr{C}_{pub}^{(t)} = \mathscr{G}(\mathbf{x}, \prod_{0\leqslant \ell\leqslant t-1}(z-g_\ell))$. We define the $(n-k)p^\lambda \times n$ matrix $\Delta_{p^\lambda}(\mathbf{H}) = (h'_{i',j'})$ by, for all $0 \leqslant i \leqslant n-k-1, 0 \leqslant j \leqslant n-1$ and $0 \leqslant \ell \leqslant p^\lambda - 1$:

$$h'_{ip^\lambda+\ell,j} = h_{i,j\ominus\ell}.$$

Let $\mathscr{C}^{(p^\lambda)}$ be the code over $\mathbb{F}_q$ with parity-check matrix $\Delta_{p^\lambda}(\mathbf{H})$. Then, $\mathscr{C}^{(p^\lambda)}$ is the QM *alternant code $\mathscr{G}(\mathbf{x}, \prod_{g\in G}(z-g))$, with automorphism group of the form $(\mathbb{Z}/p\mathbb{Z})^\lambda$.*

Remark 6.4. *$\mathscr{C}^{(p^\lambda)}$ can also be seen as the largest* QM *subcode of $\mathscr{C}_{pub}^{(t)}$ stable by $\sigma_1, \ldots, \sigma_{p^\lambda-1}$ (defined by $\sigma_\ell(i) = i \ominus_p \ell$ for $0 \leqslant i \leqslant n-1, 1 \leqslant \ell \leqslant p^\lambda - 1$).*

Proof. The proof relies on the following Lemma.

Lemma 6.5. *Let $\alpha_0, \ldots, \alpha_{\lambda-1} \in \mathbb{F}_{q^m}$ as in Theorem 5.10, generating $G = \{g_0, \ldots, g_{p^\lambda-1}\}$. Let $A = \{g_0, \ldots, g_{t-1}\}$ be a subset of G generating G Let $\mathbf{x}$ be built thanks to the α_i's according to (5.10). As a consequence, each $g \in G$ corresponds to an offset $0 \leqslant j_g < p^\lambda$ of the indices on the support such that $x_{i\oplus j_g} = x_i + g$ for all $i, 0 \leqslant i \leqslant n-1$. The public code is defined by $\mathscr{C}_{pub}^{(t)} = \mathscr{G}\big(\mathbf{x}, \prod\limits_{g\in A}(z-g)\big)$, and we define $\mathscr{C}^{(p^\lambda)} = \mathscr{G}\big(\mathbf{x}, \prod\limits_{g\in G}(z-g)\big)$. It holds that:*

$$\mathbf{c} \in \mathscr{C}^{(p^\lambda)} \iff \forall g \in G, \mathbf{c}^{\sigma_{j_g}} \in \mathscr{C}_{pub}^{(t)}.$$

Proof. Let $g \in G$ and $\mathbf{c} \in \mathscr{G}(\mathbf{x}, \Gamma)$. We introduce the polynomial syndrome associated to $\mathbf{c}$, $P_{\mathbf{c},\mathbf{x}} = \sum_{i=0}^{n-1} \frac{c_i}{z-x_i}$ (from Definition 1.22). The syndrome polynomial $P_{\mathbf{c}^{\sigma_\ell},\mathbf{x}}$ associated to $\mathbf{c}^{\sigma_\ell}$ verifies

$$P_{\mathbf{c}^{\sigma_\ell},\mathbf{x}}(z) = \sum_{i=0}^{n-1} \frac{c_{\sigma_\ell(i)}}{z-x_i} = \sum_{i=0}^{n-1} \frac{c_{i\ominus\ell}}{z-x_i} = \sum_{i=0}^{n-1} \frac{c_i}{z-x_{i\oplus\ell}} = \sum_{i=0}^{n-1} \frac{c_i}{z-g_\ell-x_i} = P_{\mathbf{c},\mathbf{x}}(z-g_\ell).$$

According to Definition 1.22, we have

$$\begin{aligned}
\mathbf{c}^{\sigma_{j_g}} \in \mathscr{C}_{pub}^{(t)} &\iff \prod_{a\in A}(z-a) | P_{\mathbf{c}^{\sigma_{j_g}},\mathbf{x}}(z) \\
&\iff \prod_{a\in A}(z-a) | P_{\mathbf{c},\mathbf{x}}(z-g) \\
&\iff \prod_{a\in A}(z+g-a) | P_{\mathbf{c},\mathbf{x}}(z) \\
&\iff \prod_{a\in A-g}(z-a) | P_{\mathbf{c},\mathbf{x}}(z)
\end{aligned}$$

As all the elements of G are pairwise distinct, the polynomials $(z-a)$ are coprime. The least common multiple of all the polynomials $\prod\limits_{a\in A-g}(z-a)$ is $P = \prod_{g\in G}(z-g)$. So, we conclude,

$$\begin{aligned}
\forall g \in G, \mathbf{c}^{\sigma_{j_g}} \in \mathscr{C}_{pub}^{(t)} &\iff \forall g \in G \prod_{A\in A-g}(z-g) | P_{\mathbf{c},\mathbf{x}}(z) \\
&\iff \textstyle\prod_{g\in G}(z-g) | P_{\mathbf{c},\mathbf{x}}(z) \\
&\iff \mathbf{c} \in \mathscr{C}^{(p^\lambda)}.
\end{aligned}$$

□

Lemma 6.5 permits to deduce a parity-check matrix of $\mathscr{C}^{(p^\lambda)}$ from any parity-check matrix $\mathbf{H}$ of $\mathscr{C}^{(t)}_{pub}$. Indeed, observe that for any row $\mathbf{h}$ of $\mathbf{H}$ and any permutation σ of the indices:

$$\begin{array}{rcl} \mathbf{c}^\sigma \in \mathscr{C}^{(t)}_{pub} & \iff & \sum_{i=0}^{n-1} c_{\sigma(i)} h_i = 0 \\ & \iff & \sum_{i=0}^{n-1} c_i h_{\sigma^{-1}(i)} = 0 \end{array}$$

To ensure that a word $\mathbf{c}$ and all its permuted words $c^{\sigma_{j_g}}$ for $g \in G$ belong to $\mathscr{C}^{(t)}_{pub}$, it suffices to permute the rows of $\mathbf{H}$ according to all the σ_g's with $g \in G$ and concatenate the obtained matrices. This is precisely what the function Δ_{p^λ} does for a group G of size p^λ. □

Thanks to Theorem 6.3, we can explain the general strategy previously presented. Using Δ_{p^λ}, we can write a new public-matrix for $\mathscr{C}^{(p^\lambda)}$ (Step 0)). Then, we can recover the multiplier vector and the support of $\mathscr{C}^{(p^\lambda)}$ (Steps 1-2-3-4-5)). Since both codes share the same support, Fact 4.1 allows to recover the multipliers of $\mathscr{C}^{(t)}_{pub}$ by solving a linear system (Step 6)).

6.2.3 Practical experiments

We considered the parameters quoted in [BCMN10, MB09, BLM11]. Remark that the public codes considered in these papers are alternant codes for $q > 2$, but (binary) Goppa codes for $q = 2$. So, we use in our experiments the systems $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}, \mathsf{McE}_{\mathbf{X},\mathbf{Y}'}$ and their reduced versions. We also generated new parameters to see how our attack scales. To compute the security levels of these new parameters with respect to ISD (Chapter 2), we used the ISDFQ software of C. Peters. [1]

Simplifying the Algebraic Systems for QM Codes. As we explained in Paragraph 6.2.1, the final folded codes have order $\widetilde{t}$ equal to p or p^2. The algebraic system $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ generated from the folded code can still be simplified by using Theorem 5.10. The folded code has a smaller but non-trivial automorphism groups. Consequently, we can use the linear relations (5.10) and (6.1) to decrease the number of variables. From now on, we will always assume that such linear relations are used in our systems.

Signature Schemes – Experimental Results. The QM codes used in signature schemes always have the maximal possible lengths, namely $n = q^m - t$. This leads to codes of relatively big lengths. The total numbers of variables in the systems $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ can be also huge, typically up to $\geqslant 500$. However, $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ contains very few variables compared to the number of equations. For example, $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ contains 19671 equations and 21 variables for $(q = 3, m = 11)$. To compare both approaches on a tiny example, we picked a code with parameters $(q = 16, m = 3)$, length $n = 360$ and block size $t = 8$. Solving $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ (containing 126 equations in 49 variables) took 311 seconds, whereas $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ contains 84 equations in only 8 variables and was solved in 0.01 second. Thus, we choose to perform the structural elimination in all possible cases. This permitted to mount a key-recovery for almost all the signature parameters proposed in [BCMN10, BLM11]. In Table 6.1, we detail our experimental results. We have used MAGMA [BCP97b] (V2.17-1) to implement our attack. All the timings have been obtained on a 2.93 GHz Intel®. The Gröbner bases computation in MAGMA are performed with an optimized version of F_4 [Fau99]. We have been able to break all the parameters proposed in [BCMN10, BLM11]. For most parameters proposed in [BLM11], we have been

1. http://christianepeters.wordpress.com/publications/tools/

able to recover the secret in few seconds (the last row took less than 2 hours). As a conclusion, the use of QM codes introduces a fatal weakness in the signature context. In view of our new attack, it seems impossible to find secure parameters for QM (resp. QD) signature schemes.

Solving $\mathsf{elimMcE}_{\mathbf{X}',\mathbf{Y}'}$ (with $\widetilde{t} = p$).

$p = 2$	m	n	t	$\widetilde{t}$	n_0	$(2m-2)$ unk.	$2(n_0 - m)$ eq.	Bi-degree	MAGMA	Sec. level
2	13	8176	16	2	511	25	996	$(2,1),(2,2)$	1.9 s.	84
2	13	8176	15	2	511	25	996		1.9 s.	81
2	14	16368	14	2	1023	27	2018		3.0 s.	84
2	14	16368	13	2	1023	27	2018		3.0 s.	81
2	15	32752	12	2	2047	29	4064		5.4 s.	82

Solving $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ (with $\widetilde{t} = p^2$).

$p > 2$	m	n	t	$\widetilde{t}$	n_0	$(2m-1)$ unk.	$(n_0 - m)$ eq.	Bi-degree (p,p)	MAGMA	Sec. level
3	11	177048	9	9	19682	21	19671	$(3,3)$	3.4 s.	80
5	8	390495	15	25	15624	15	15616	$(5,5)$	82 s.	128

Solving $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ when $t = \widetilde{t} = p$ (so, without folding).

$p > 2$	m	n	t	$\widetilde{t}$	n_0	# unk.	$\leqslant (p-1)t$ eq.	Bi-degree	MAGMA	Sec. level
13	4	28509	13	13	2196	12	8	$(1,i), 1 \leqslant i \leqslant 7$	0.05 s	80
13	5	371228	13	13	28560	12	10	$(1,i), 1 \leqslant i \leqslant 7$	5873 s.	112

Table 6.1 – Practical attacks against signature schemes with parameters from [BCMN10, BLM11].

For a public alternant code with $t = p$, neither the folding nor the structural elimination are possible (as explained in 4.2). For those cases, we pick equations from the first block of rows. That is:

$$\sum_{j=0}^{n-k-1} a_{i,j} Y_j \left(X_j^{\ell} - X_{n-k+i}^{\ell} \right) = 0,$$

with $i \in \{0, \ldots, t-1\}$ (instead of $0 \leqslant i \leqslant k-1$). This system only involves X_0, X_1, and $X_t, X_{2t}, \ldots, X_{(n_0-m)t}$, contains up to $t(p-1)$ equations and $2m-1$ variables. The strategy was very successful for small m's ($m \leqslant 6$).

Encryption – Experimental Results. The algebraic systems are harder to solve for encryption parameters. In addition of MAGMA, we also use FGB to compute the Gröbner bases. For FGB, we reported the basic number of expected operations. To estimate this number, we have mixed exhaustive search and Gröbner bases. This explains that we reported number of operations $\geqslant 2^{80}$. To see how the attack behaves, we also generated ourselves cryptographic secure parameters with ISDFQ (marked with a $\star$ in the table below). Our practical results are somewhat orthogonal with ISD. In most cases, the complexity of ISD grows whilst the efficiency of our attack increases. This is due to the fact that the folding process makes the value of t irrelevant for our attack; which is not the case for ISD. For encryption parameters, the choice between $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ and $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ is less clear than in the signature context. Both systems have a reduced number of equations compared to the signature case. Thus, the higher degree induced by the structural elimination may be an obstacle to the resolution of $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$. Solving this system proved to be very efficient for codes over $\mathbb{F}_q$ with $q = 2^s$ and

$s > 1$, as we found multipliers and support up to 45000 times faster than FOPT's attack in [FOPT10a], but is less efficient for $p \geqslant 3$. We mention that some parameters proposed could not be solved in practice. It is not unlikely that better results could be obtained in the future by using a specific Gröbner strategy for solving the systems (which are still structured). In any case, we see the experimental results as a practical validation of the structural weakness of compact codes coming from the folding process.

Solving $\mathsf{elimMcE}_{\mathbf{X}',\mathbf{Y}'}$ with $\widetilde{t} = p$.

$q = 2$	m	n	t	$\widetilde{t}$	n_0	$2m-1$ unk.	$2(n_0-m)$ eq.	Magma	F_5/FGb	FOPT	Sec. level
2	16	4864	2^5	2	152	31	272	18 s.		N.A.	128
2	12	3200	2^7	2	25	23	22		$\leqslant 2^{83.5}$ op.	N.A.	128
2	14	5376	2^7	2	42	27	36		$\leqslant 2^{96.1}$ op.	N.A.	226
2	15	11264	2^9	2	22	29	14		$\leqslant 2^{146}$ op.	N.A.	256
2	16	6912	2^8	2	27	31	33		$\leqslant 2^{168}$ op.	N.A.	218
2	16	8192	2^8	2	32	31	32		$\leqslant 2^{157}$ op.	N.A.	256

Solving $\mathsf{elimA}_{\mathbf{X}',\mathbf{Y}'}$ with $\widetilde{t} = p^2$.

$q > 2$	m	n	t	$\widetilde{t}$	n_0	$2m-1$ unk.	(n_0-m) eq.	Magma	F_5/FGb	FOPT	Sec. level
2^4	4	2048	2^6	4	32	8	28	0.01 s.		0.50 s	128
2^4	4	4096	2^7	4	32	8	28	0.01 s.		7.1 s	128
2^2	8	3584	2^6	4	56	15	48	0.04 s.		1,776 s	128
3	8	3645	3^4	9	45	15	37		$\leqslant 2^{44.5}$ op.	N.A.	224 $\star$
3	11	4860	3^4	9	60	21	49		3 d. 17h.	N.A.	192 $\star$
3	11	6885	3^4	9	85	21	64		181 s.	N.A.	261 $\star$
5	8	2500	5^2	5^2	100	15	92		2.9s	N.A.	107 $\star$
5	8	1375	5^2	5^2	55	15	47		160s	N.A.	85 $\star$
5	9	1750	5^2	5^2	70	17	61		728.4s	N.A.	89.8 $\star$
5	10	2000	5^2	5^2	80	19	70		5941.9s	N.A.	91.3 $\star$

Solving $\mathsf{A}_{\mathbf{X},\mathbf{Y}'}$ with $t = \widetilde{t} = p$.

q	m	n	t	$\widetilde{t}$	n_0	# unk.	$\leqslant t(p-1)$ eq.	Magma	FOPT	Sec. level
167	3	668	167	167	4	5	7	0.020 s.	N.A.	80
241	3	964	241	241	4	5	7	0.020 s.	N.A.	112
41	3	451	41	41	11	14	24	0.030 s.	N.A.	80
5	5	1000	5^3	5	8	11	12	140 s.	N.A.	80
11	5	1089	11^2	11	9	12	20	225 s.	N.A.	112
7	5	735	7^2	7	15	18	60	900 s.	N.A.	80
7	6	1813	7^2	7	37	41	186	60 s.	N.A.	128

Table 6.2 – Practical attacks against the encryption parameters proposed in [MB09, BLM11]. N.A. means that the parameters could not be addressed in [FOPT10a].

6.3 Conclusion about the symmetric Alternant Codes

Our attack proved to be a fatal weakness for the signature schemes using symmetric alternant codes. In those cases, the considerable amount of equations allowed a very fast resolution. This is linked with the fact that the codes required to perform a CFS signature have a very high rate. For the encryption parameters, the rates of the code are lower. This entails lower amounts of equations and systems more difficult to solve. This could suggest to instantiate McEliece schemes with symmetric alternant codes of low rates. The problem with such codes is that, regarding message-recovery, the ISD-based attacks become easier when the

rate is low. We provide in Figure 6.2 a comparison of the complexities of our key-recovery method and message-recovery for QM codes defined over $\mathbb{F}_5$, with extension degree $m = 5$, block size $t = 5^2$, and rates $1 - \frac{m}{n_0}$. We observe that no value of n_0 guarantees a security above 2^{80} against both message and key-recovery.

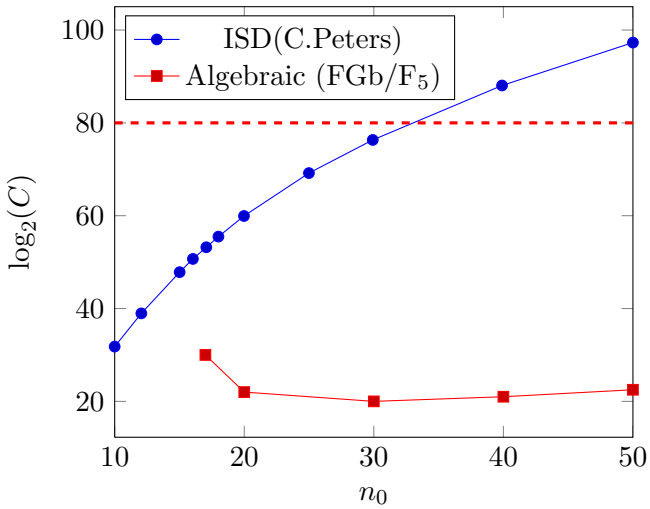


Figure 6.2 – Comparison of the complexities of ISD (message-recovery) and algebraic attacks (key-recovery) against QM codes over $\mathbb{F}_5$ with $m = 5$, $t = 25$, $n = n_0 t$.

More generally, the folding construction has to be taken into account as soon as the public code is an alternant code with QC or QM structure. We saw already several examples that motivated these chapters, but we can also mention the proposal of Persichetti [Per12a]. In this case, the public code belongs to a specific family of alternant codes (the *Srivastava* codes). The folding construction and the alternant modelling both apply. An open question is then to use this to design an attack against McEliece schemes using QD Srivastava codes (as proposed in [Per12a]).

Chapter 7

Algebraic cryptanalysis of Wild McEliece

This chapter presents results from the article *Algebraic Cryptanalysis of McEliece with Goppa polynomials of a special form* published at Asiacrypt 2014 with Jean-Charles Faugère and Ludovic Perret [FPdP14].

It is dedicated to a recently proposed variant of McEliece schemes using, as in Chapter 6, a particular subclass of Goppa codes (Definition 1.22 of Chapter 1). This variant, proposed by Bernstein, Lange and Peters in [BLP11b, BLP11c], uses *Wild Goppa codes*, that is Goppa codes over $\mathbb{F}_q, q > 2$, with a Goppa polynomial of the form $f(z)g(z)^{q-1}$.

Definition 7.1 (Wild Goppa/Masked Wild Goppa)**.** *Let* $\mathbf{x}$ *be an* n*-tuple* $(x_0, \ldots, x_{n-1})$ *of distinct elements of* $\mathbb{F}_{q^m}$*. Let* $g(z) \in \mathbb{F}_{q^m}[z]$ *(resp.* $f(z) \in \mathbb{F}_{q^m}[z]$*) be a squarefree polynomial of degree* t *(resp.* u*) satisfying* $g(x_i) \neq 0$ *(resp.* $f(x_i) \neq 0$*) for all* $i, 0 \leqslant i \leqslant n-1$*. A* Wild Goppa code *is a Goppa code whose Goppa polynomial is of the form* $\Gamma(z) = g(z)^{q-1}$*. A* Incognito Wild Goppa code *is a Wild Goppa code whose Goppa polynomial is such that* $\Gamma(z) = f(z)g(z)^{q-1}$*.*

The reason for using those Goppa polynomials lies in the following result.

Theorem 7.2 ([SKHN76])**.** *Let the notations be as in Definition 7.1. It holds that*

$$\mathscr{G}(\mathbf{x}, f(z)g^{q-1}(z)) = \mathscr{G}(\mathbf{x}, f(z)g^q(z)). \tag{7.1}$$

Thus, the code $\mathscr{G}(\mathbf{x}, f(z)g^q(z))$ *has dimension* $\geqslant n - m\left((q-1)t + u\right)$ *and minimum distance* $\geqslant qt + u$*.*

This is a generalization of a well-known property for $q = 2$. The advantage of Wild Goppa codes compared to standard Goppa codes is that $\lfloor (qt+u)/2 \rfloor$ errors can be decoded efficiently (instead of $\lfloor ((q-1)t+u)/2 \rfloor$ as in Paragraph 1.3 of Chapter 1) for the same code dimension ($n - ((q-1)t+u)m$ in most cases). This increases the difficulty of the syndrome decoding problem (Chapter 2). Hence, to design a McEliece-like scheme of a given level of security, Wild Goppa codes induce smaller keys than binary Goppa codes (for details, see [BLP11b, Section 7] and [BLP11c, Section 5]).

In this chapter, we present a new algebraic modelling dedicated to Wild Goppa codes and more generally to Goppa codes whose Goppa polynomials have multiple factors. In Section 7.1, we reduce the key-security of a Wild McEliece scheme to the difficulty of solving an algebraic system which contains contains fewer variables than the FOPT modelling. We show that to break a Wild McEliece scheme it suffices to solve (roughly)

$$\mathcal{W}(\mathbf{Z}) = \bigcup_{1 \leqslant u \leqslant q} \Big\{ g_{i,0} Z_0^u + \cdots + g_{i,n-1} Z_{n-1}^u = 0 \ \mid \ i \in \{0, \ldots, k-1\} \Big\}, \tag{7.2}$$

with unknowns $Z_0, \ldots, Z_{n-1}$ and $\mathbf{G} = (g_{i,j})_{\substack{0 \leqslant i \leqslant k-1 \\ 0 \leqslant j \leqslant n-1}}$ a generator matrix of the public code. First, we prove a very unexpected property on the solutions of (7.2): they form a vector space. This helps us in the resolution, because it means that many Z_j's can be fixed arbitrarily. Second, this solution vector space is nothing but a GRS code with same private elements as the public code, so that the famous Sidelnikov-Shestakov attack ([SS92], recalled in Paragraph 2.3.1) yields the private elements from a basis of the solutions of $\mathcal{W}(\mathbf{Z})$.

We improve this method in Section 7.2. We describe how to solve $\mathcal{W}(\mathbf{Z})$, which implies to compute a Gröbner basis (Chapter 3), and how to deduce the private key, which requires an essentially polynomial-time treatment. Figure 7.1 (on page 119) sums up all the steps of the key recovery algorithm. It can be applied as soon as the Goppa polynomial has a multiple factor.

We applied this strategy to the cryptanalysis of Wild McEliece in Section 7.3. We used specific features of some Wild Goppa codes (defined over $\mathbb{F}_q$ with $q = p^s$ with $s > 1$), to fix more variables. We ended up with a polynomial system containing, on some examples, as few as 10 variables. We review the practical attacks that we carried out, which included challenges generated by the designers of Wild McEliece with security level up to 2^{128}.

$$\mathcal{W}_{q,a}(\mathbf{Z}) = \bigcup_{u \in \mathcal{P}_a} \left\{ \sum_{j=0}^{n-1} g_{i,j} Z_j^u = 0 \quad | \quad 0 \leqslant i \leqslant k-1 \right\}$$

We solve the system several times by fixing many variables.

$$\left\{ \begin{array}{ll} \textsc{Solve}(\mathcal{W}_{q,a}(\mathbf{Z}) \bigcup \{\mathbf{Z} = (1, \ldots, 0, Z_{d_a}, \ldots, Z_{n-1})\}) & \to \mathbf{v}^{(0)} \\ \vdots & \vdots \\ \textsc{Solve}\ (\mathcal{W}_{q,a}(\mathbf{Z}) \bigcup \{\mathbf{Z} = (0, \ldots, 1, Z_{d_a}, \ldots, Z_{n-1})\}) & \to \mathbf{v}^{(d_a-1)} \end{array} \right.$$

We perform the Fröbenius alignement to have a cleaner vector space (i.e. same Fröbenius power on all the solutions).

$$\begin{aligned} \mathscr{C}_\Sigma &= \mathrm{Span}\left(\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(d_a-1)}\right) \\ &= \sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell) \end{aligned}$$

We perform a suitable intersection $\mathscr{C}_\Sigma \cap (\mathscr{C}_\Sigma)^{p^{s-a}}$ to recover a single GRS.

$$\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})^{p^{s-a}}$$

We adapt Sidelnikov-Shestakov attack to recover the secret key.

Secret $\mathbf{x}, \mathbf{y}, g(z)$

For Incognito, a extra linear algebra step allows to recover the last part of the secret key.

Secret $f(z)$ for incognito

Figure 7.1 – Overview of our attack against Wild McEliece.

7.1 New Algebraic Modelling with a Vector Space Structure on the Zero Set

As Wild Goppa codes are in particular Goppa codes, the same algebraic modelling as in [FOPT10a] holds. However, we show that the multiplicity in the Goppa polynomial of a Wild code allows to write an algebraic system whose solution set $\mathcal{S}$ has a very surprising structure (Definition 7.5). Indeed, it appears that $\mathcal{S}$ includes the union of several vector spaces. The vector spaces correspond in fact to sums of GRS codes (Definition 1.17) which have almost the same support $\mathbf{x}$ and multiplier vector $\mathbf{y}$ as the public-key of the attacked Wild McEliece Incognito scheme (Theorem 7.6).

7.1.1 First Simplification of the System (FOPT)

We give in this paragraph the intuition leading to the central Definition 7.5 and Theorem 7.6. This can be seen as a first version of the attack, with some details to be given only in the sequel. Let $\mathscr{C} = \mathscr{G}(\mathbf{x}, g^{q-1}(z))$ be an $[n,k]_q$ Wild Goppa code with generator matrix $\mathbf{G} = (g_{i,j})_{\substack{0\leqslant i\leqslant k-1\\ 0\leqslant j\leqslant n-1}}$. The multipliers associated to $\mathscr{C}$ (seen as an alternant code) are defined by $\mathbf{v} = 1/g(\mathbf{x})^{q-1}$. The modelling from [FOPT10a] would introduce unknowns X_i and V_i for the support and multipliers respectively, and write the equations

$$\Big\{ g_{i,0}V_0X_0^{\ell} + \cdots + g_{i,n-1}V_{n-1}X_{n-1}^{\ell} = 0 \ \mid \ i \in \{0,\ldots,k-1\}, \ell \in \{0,\ldots,qt-1\} \Big\}.$$

The first crucial remark is that many more equations can be written, thanks to the easy inclusion $\mathscr{C} \subseteq \mathscr{G}(\mathbf{x}, g^u(z))$ for any $1 \leqslant u \leqslant q-1$. This comes from the fact that, for a codeword $\mathbf{m} \in \mathscr{C}$, it holds thanks Definition 1.22 that

$$g^{q-1}(z) | \sum_{i=0}^{n-1} \frac{m_i}{z-x_i} \implies g^u(z) | \sum_{i=0}^{n-1} \frac{m_i}{z-x_i}.$$

Therefore, $\mathbf{V}_{ut}(\mathbf{x}, \mathbf{v}^{(u)}) \times \mathbf{G}^T = (\mathbf{0})_{ut\times k}$ for any $1 \leqslant u \leqslant q-1$, with $\mathbf{v}^{(u)} = (g(\mathbf{x}))^{-u}$. Here, we remark that we can write simultaneously *all* these equations and introduce only one vector of unknowns $\mathbf{y} = g(\mathbf{x})^{-1} = \mathbf{v}^{(1)}$ instead of $q-1$ vectors $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(q-1)}$, since $\mathbf{v}^{(u)} = \left(\mathbf{v}^{(1)}\right)^u$ (by this we mean the exponentiation component-wise). Therefore, we obtain the system

$$\bigcup_{1\leqslant u\leqslant q} \Big\{ g_{i,0}Y_0^uX_0^{\ell_u} + \cdots + g_{i,n-1}Y_{n-1}^uX_{n-1}^{\ell_u} = 0 \ \mid \ i \in \{0,\ldots,k-1\}, \ell_u \in \{0,\ldots,ut-1\} \Big\} \quad (7.3)$$

In essence, this proves the following lemma (in which we included equations related to the incognito polynomial)

Lemma 7.3. *Let $\mathbf{G}_{pub}$ be a generator matrix of $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}, f(z)g^{q-1}(z))$ a Wild Incognito Goppa code, $\mathbf{y} = g(\mathbf{x})^{-1}$, $\mathbf{w} = f(\mathbf{x})^{-1}$ and $t = \deg(g(z))$. We have:*

$$\bigcup_{u_y=0}^{q} \Bigg\{ \sum_{j=0}^{n-1} g_{i,j} \left(w_jx_j^u\right)^b y_j^{u_y} x_j^{u_x} = 0 \ \mid \ \begin{array}{l} 0 \leqslant i \leqslant k-1, 0 \leqslant u_x \leqslant u_yt-1, \\ 0 \leqslant u \leqslant \deg(f)-1, b \in \{0,1\}, (b,u_y) \neq (0,0) \end{array} \Bigg\}.$$

Proof. The only difference with system (7.3) is the factor $f(z)$ in the Goppa polynomial. For all $0 \leqslant u_y \leqslant q$ and $0 \leqslant b \leqslant 1$ (and $(u_y, b) \neq (0,0)$), it holds that

$$\mathscr{C}_{pub} \subseteq \mathscr{G}(\mathbf{x}, f^b(z) g^{u_y}(z)).$$

As $\mathscr{G}(\mathbf{x}, f^b(z) g^{u_y}(z))$ has parity check matrix $\mathbf{V}_{d_{tot}}(\mathbf{x}, \mathbf{w}^b \mathbf{y}^{u_y})$ (with $d_{tot} = b \deg(f) + u_y t$), the matrix products $V_{d_{tot}}(\mathbf{x}, \mathbf{w}^b \mathbf{y}^{u_y}) \times \mathbf{G}_{pub}^T = \mathbf{0}_{d_{tot} \times k}$ yield all the relations of the lemma. □

System (7.3) contains q times more equations than in [FOPT10a], but still as many variables. However, by setting $\ell_u = 0$ in all the equations, we obtain a much simpler subset of equations where the X_i's have disappeared, so that one could hope to find efficiently the y_i's (and the x_i's would then be easy to recover thanks to Lemma 4.1):

$$\bigcup_{1 \leqslant u \leqslant q} \Big\{ g_{i,0} Y_0^u + \cdots + g_{i,n-1} Y_{n-1}^u = 0 \ \mid \ i \in \{0, \ldots, k-1\} \Big\}. \tag{7.4}$$

Unfortunately, except for very specific cases (Wild Goppa codes with $t = 1$), it is hopeless to find directly the y_i's with System (7.4). The reason is that System (7.4) has *infinitely* many solutions (as soon as $t > 1$). Indeed, notice that when substituting a vector of the form

$$\mathbf{y}\mathbf{x}^\ell = (y_0 x_0^\ell, \ldots, y_{n-1} x_{n-1}^\ell)$$

in (7.4), we obtain exactly equations of Lemma 7.3 (with $u_y = u, u_x = \ell u, b = 0$), so that we know that all the $\mathbf{y}\mathbf{x}^\ell$ for $0 \leqslant \ell \leqslant t-1$ are solutions. Moreover, by applying a Frobenius endomorphism $x \mapsto x^{q^e}$ we see that all the $\mathbf{y}^{q^e} \mathbf{x}^{\ell q^e}$ are also solutions. This proves that the vector $\mathbf{y}$ is not the only solution, but provides actually a finite number of solutions. We obtain an infinity of solutions when we remark that the solutions $\mathbf{y}\mathbf{x}^\ell$ can be *linearly combined*. In other words, the solutions contain a *vector space*. This will be proved along with our central Theorem 7.6.

For now, we explain the consequences of this fact. In order to have efficient polynomial system solving algorithms, it is necessary to transform the system (7.4) so that it has a *finite number of solutions*. To do so, thanks to the vector space structure, we know that we can fix arbitrarily a number of coordinates of the solution vector equal to the dimension of the solution vector space. Note that this is very favorable for the polynomial system resolution, since it eliminates variables. Once, we have obtained a finite number of solutions, the central question is : *how can we recover any secret element of $\mathscr{C}$ from those solutions ?*

When having a closer look at the solutions, we realize that what we obtain is an *unknown* linear combination of the solutions $\mathbf{y}\mathbf{x}^\ell$ with $0 \leqslant \ell \leqslant t-1$ (or their Frobenius $\mathbf{y}^{q^e} \mathbf{x}^{\ell q^e}$), that is a vector with coordinates of the form

$$\left(\alpha_0 y_0 + \alpha_1 y_0 x_0 + \cdots + \alpha_{t-1} y_0 x_0^{t-1}, \ldots, \alpha_0 y_{n-1} + \alpha_1 y_{n-1} x_{n-1} + \cdots + \alpha_{t-1} y_{n-1} x_{n-1}^{t-1}\right)^{q^e}.$$

For now, we suppose $e = 0$ to simplify, and we set $Q = \alpha_{t-1} z^{t-1} + \cdots + \alpha_0$, we see that a solution can be written

$$(y_0 Q(x_0), \ldots, y_{n-1} Q(x_{n-1})).$$

In other words, the solutions are codewords of the code $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$. The strategy (detailed in Section 7.2) will consist in finding a generator matrix of $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$, and perform a Sidelnikov-Shestakov attack (recalled in Paragraph 2.3.1) to recover $\mathbf{x}$ and $\mathbf{y}$. We sum up the central tool of this first cryptanalysis:

Theorem 7.4. *Let $q = p^s$ (p prime and $s \geqslant 0$). Let $\mathbf{G}_{pub} = (g_{i,j})_{\substack{0\leqslant i\leqslant k-1\\ 0\leqslant j\leqslant n-1}}$ be a generator matrix of a Wild Goppa code $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}, g^{q-1}(z))$. Let $\mathbf{y} = g(\mathbf{x})^{-1}$. Then, the solutions $\mathcal{S}$ of the system*

$$\bigcup_{1\leqslant u\leqslant q} \left\{ g_{i,0}Z_0^u + \cdots + g_{i,n-1}Z_{n-1}^u = 0 \ \mid \ i \in \{0, \ldots, k-1\} \right\}$$

contain the union of m GRS *codes:*

$$\bigcup_{0\leqslant e\leqslant m-1} \left(\mathrm{GRS}_t(\mathbf{x}^{q^e}, \mathbf{y}^{q^e})\right) \subseteq \mathcal{S}.$$

This theorem will be proven along with its refinement (Theorem 7.6). Note that we only need the multiplicity in the polynomial $g(z)^{q-1}$ to establish the GRS structure of the solutions. As a consequence, if System (7.4) can be solved efficiently, the key recovery on a McEliece scheme with multiplicity in the Goppa polynomial boils down to that of GRS code, which is known to be weak.

7.1.2 The Algebraic Modelling $\mathcal{W}_{q,a}(\mathbf{Z})$

We explained that, when solving System (7.4), unknowns could be fixed arbitrarily thanks to the vector space structure of the solution set. Following this idea, we introduce an algebraic system which is a *subset* of the System (7.4). By removing equations from (7.4), we naturally introduce more solutions. However, the subset is chosen so as to:

— keep the vector space structure of the solutions,

— raise its dimension.

The purpose is to fix more coordinates (that is, more variables) of the unknowns' vector.

Definition 7.5. *Let $q = p^s$ (p prime and $s \geqslant 0$). Let $\mathbf{G}_{pub} = (g_{i,j})_{\substack{0\leqslant i\leqslant k-1\\ 0\leqslant j\leqslant n-1}}$ be a generator matrix of a masked Wild Goppa code $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}, f(z)g^{q-1}(z))$. For an integer $a, 0 < a \leqslant s$, we define the system $\mathcal{W}_{q,a}(\mathbf{Z})$ as follows :*

$$\mathcal{W}_{q,a}(\mathbf{Z}) = \bigcup_{u\in\mathcal{P}_a} \left\{ \sum_{j=0}^{n-1} g_{i,j}Z_j^u = 0 \quad \mid \quad 0 \leqslant i \leqslant k-1 \right\} \tag{7.5}$$

with $\mathcal{P}_a = \{1, 2, \ldots, p^a - 1\} \cup \{p^a, p^{a+1}, \ldots, q\}$.

The parameter a in $\mathcal{P}_a$ determines the exponents considered for the Z_j's in the system (7.5). For $a = s$, we consider all the powers Z_j^u where u ranges in $\{1, \ldots, q\}$ and we recover System (7.4). Removing some exponents leads to a system with fewer equations and may seem counter-intuitive at first sight (the more equations, the better it is for solving a polynomial system). However, as we explained, the situation is different here due to the specific structure of the solutions of $\mathcal{W}_{q,a}(\mathbf{Z})$, since we will be able to fix more variables. The counterpart is that the solutions do no longer yield a GRS code but a *sum* of GRS codes.

Theorem 7.6. *Let the notations be as in Definition 7.5. Let $\mathbf{y} = g(\mathbf{x})^{-1}$, and $t = \deg(g)$. The solutions $\mathcal{S}$ of $\mathcal{W}_{q,a}(\mathbf{Z})$ contain the union of m vector spaces which are sums of* GRS *codes, that is:*

$$\bigcup_{0\leqslant e\leqslant m-1} \left(\sum_{\ell\in\mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^{\ell q^e}, \mathbf{y}^{\ell q^e}) \right) \subseteq \mathcal{S} \text{ where } \mathcal{L}_a = \bigcup_{0\leqslant r\leqslant s-1-a} \{p^r, 2p^r, \ldots, (p-1)p^r\} \cup \left\{p^{s-a}\right\}.$$

In Theorem 7.6, we prove an inclusion. In practice, as the system is highly overdefined, we always observed that this subset was *all* the solutions.

Proof. First, we give the global idea of the proof. The goal is to show that the elements of $\sum_{\ell\in\mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^{\ell q^e}, \mathbf{y}^{\ell q^e})$ are solutions of $\mathcal{W}_{q,a}(\mathbf{Z})$. We can assume that $e = 0$ w.lo.g.

Let $\mathbf{z} = (z_1, \ldots, z_n) \in \sum_{\ell\in\mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)$. We write the coordinates of $\mathbf{z}$ as $z_j = \sum_{\ell\in\mathcal{L}_a} y_j^\ell Q_\ell(x_j^\ell)$, where the Q_ℓ's are polynomials of degree $\leqslant t-1$ in $\mathbb{F}_{q^m}[z]$. We have to prove that

$$\sum_{j=0}^{n-1} g_{i,j} z_j^u = 0 \text{ for } u \in \mathcal{P}_1 \cup \mathcal{P}_2, \text{where } \mathcal{P}_1 = \{1, 2, \ldots, p^a - 1\} \text{ and } \mathcal{P}_2 = \{p^a, \ldots, p^s\}.$$

The idea is to develop $z_j^u = \left(\sum_{\ell\in\mathcal{L}_a} y_j^\ell Q_\ell(x_j^\ell)\right)^u$ with Newton multinomial. The development is performed slightly differently whether $u \in \mathcal{P}_1$ or $u \in \mathcal{P}_2$ (see details below). In both cases, we end up with a result of the form $z_j^u = \sum_{u_x,u_y} \alpha_{u_x,u_y} y_j^{u_y} x_j^{u_x}$, so that our sum writes:

$$\sum_{j=0}^{n-1} g_{i,j} z_j^u = \sum_{u_x,u_y} \alpha_{u_x,u_y} \sum_{j=0}^{n-1} g_{i,j} y_j^{u_y} x_j^{u_x}.$$

We set $b = 0$ in the system of Lemma 7.3. We obtain $\sum_{j=0}^{n-1} g_{i,j} y_j^{u_y} x_j^{u_x} = 0$ for (u_y, u_x) such that $1 \leqslant u_y \leqslant t$ and $0 \leqslant u_x \leqslant u_y t - 1$. Thus to conclude that $\sum_{j=0}^{n-1} g_{i,j} z_j^u = 0$, we check that all the couples (u_x, u_y) appearing in the sum satisfy those conditions.

Detailed Multinomial Development. We give the multinomial development of the $z_j^u = \left(\sum_{\ell\in\mathcal{L}_a} y_j^\ell Q_\ell(x_j^\ell)\right)^u$ and conclude the proof separately for $u \in \mathcal{P}_1$ and $u \in \mathcal{P}_2$.

Case $u \in \mathcal{P}_1$. We pick $u \in \{1, 2, \ldots, p^a - 1\}$ and use the multinomial formula to expand $\left(\sum_{\ell\in\mathcal{L}_a} y_j^\ell Q_\ell(x_j^\ell)\right)^u$. Namely, with $L_a = \#\mathcal{L}_a$, we have:

$$\left(\sum_{\ell\in\mathcal{L}_a} y_j^\ell Q_\ell(x_j^\ell)\right)^u = \sum_{\substack{0\leqslant u_1,\ldots,u_L\leqslant u \\ u_1+\cdots+u_L=u}} \left(\binom{u}{u_1,\ldots,u_L} y_j^{\left(\sum_{\ell\in\mathcal{L}_a} \ell u_\ell\right)} \prod_{\ell\in\mathcal{L}_a} Q_\ell(x_j^\ell)^{u_\ell} \right).$$

Let's look at each term $y_j^{u_y} x_j^{u_x}$ in the sum. For $u_1, \ldots, u_L$ non-negative integers with $u_1 + \cdots + u_L = u$, it holds that $u_y = \sum_{\ell\in\mathcal{L}_a} \ell u_\ell \leqslant \max(\mathcal{L}_a) \sum_{\ell\in\mathcal{L}_a} u_\ell \leqslant p^{s-a} u \leqslant p^s$. For each $y_j^{u_y}$, several terms $y_j^{u_y} x_j^{u_x}$ appear after expanding $\prod_{\ell\in\mathcal{L}_a} Q_\ell(x_j^\ell)^{u_\ell}$. In $Q_\ell(x_j^\ell)^{u_\ell}$ the maximal power u_x appearing is $\ell u_\ell(t-1)$ (as Q_ℓ has degree $t-1$). Thus, in $\prod_{\ell\in\mathcal{L}_a} Q_\ell(x_j^\ell)^{u_\ell}$, the maximal power is $(t-1)\sum_{\ell\in\mathcal{L}_a} \ell u_\ell = (t-1)u_y \leqslant t u_y - 1$.

Case $u \in \mathcal{P}_2$. We pick $b \in \{a, \ldots, s\}$. Then, thanks to Lemma 7.7 (proved below),

$$z_j^{p^b} = \left(\sum_{\ell\in\mathcal{L}_a} y_j^\ell Q_\ell(x_j^\ell)\right)^{p^b} = \sum_{\ell\in\mathcal{L}_a} y_j^{\ell p^b} Q_\ell^{(b)}(x_j^{\ell p^b}).$$

Pick $\ell \in \mathcal{L}_a$, it writes $\ell = \alpha p^c$ with $1 \leqslant \alpha < p$ and $0 \leqslant c \leqslant s-a$. Thus we have $\ell p^b = \alpha p^{c+b}$. The euclidian division of $c+b$ by s gives $c+b = ds+e$ with $0 \leqslant e < s$. The exponent ℓp^b then writes $\ell p^b = \alpha p^e p^{ds} = \alpha p^e q^d$. As $g_{i,j}^q = g_{i,j}$ it holds that $\left(\sum_{j=0}^{n-1} g_{i,j} y_j^{\alpha p^e q^d} Q_\ell^{(b)}(x_j^{\alpha p^e q^d})\right)^{q^m/q^d} = \sum_{j=0}^{n-1} g_{i,j} y_j^{\alpha p^e} Q_\ell^{(b-ds)}(x_j^{\alpha p^e})$. As the $R_\ell^{(b-ds)}$'s have degree lower than t, all the terms of the sum are of the form $y_j^{u_y} x_j^{u_x}$ with $u_y \leqslant q$ (since $\alpha p^e < p^s$) and $u_x \leqslant u_y t - 1$. This terminates the proof. □

For completeness, we state (and prove) the technical Lemma 7.7.

Lemma 7.7. *Let $q = p^s$ (p prime and $s > 0$), and $Q = \gamma_t z^t + \cdots + \gamma_0 \in \mathbb{F}_{q^m}[z]$ be a polynomial of degree t. For all j, it holds that:*

$$\begin{aligned} Q(z)^{p^j} &= \gamma_t^{p^j}(z^t)^{p^j} + \cdots + \gamma_0^{p^j} \\ &= \gamma_t^{p^j}(z^{p^j})^t + \cdots + \gamma_0^{p^j} \\ &= Q^{(j)}(z^{p^j}). \end{aligned}$$

where $Q^{(j)} = \gamma_t^{p^j} z^t + \cdots + \gamma_0^{p^j}$ is the polynomial of same degree as Q obtained by raising all the coefficients to the p^j-power.

7.2 Solving the Goppa Decoder Recovery Problem for codes with multiplicities

We explain now how to recover a private decoder from the equations $\mathcal{W}_{q,a}(\mathbf{Z})$. That is, we want to find a secret support $\mathbf{x}'$, a correct $g'(z)$ and a correct $f'(z)$ in the incognito case.

The first step consists in exploiting the structure of the solutions of $\mathcal{W}_{q,a}(\mathbf{Z})$ to speed-up its resolution (Paragraph 7.2.1). We have to solve it several times. Then, we extract the basis of a GRS code from the solutions obtained at the previous steps. This is done in Paragraph 7.2.2. Then, we show in Paragraph 7.2.3 how to recover a private support $\mathbf{x}$ and Goppa polynomial $g(z)$ of the masked Wild Goppa code. This is the full description of a plain Wild Goppa code. In the Incognito case ($\deg(f) > 0$), we explain in Paragraph 7.2.4 that an extra linear step enables to find f.

Figure 7.1 provides a diagram which recapitulates all the steps performed to recover the secret key.

7.2.1 Boosting the Resolution of $\mathcal{W}_{q,a}(\mathbf{Z})$

First, we use the particular structure of the solution set of the non-linear system $\mathcal{W}_{q,a}(\mathbf{Z})$ to solve it. When looking for a vector in a subspace of $\mathbb{F}_{q^m}^n$ of dimension d, then you can safely fix d coordinates arbitrarily and complete the $n-d$ so as to obtain a vector of this subspace. This corresponds to computing intersections of the subspace with d hyperplanes. With this idea, we deduce the following corollary of Theorem 7.6.

Corollary 7.8. *Let $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}, f(z)g^{q-1}(z))$ be a masked Wild Goppa code. Let $t = \deg(g)$, $\mathcal{W}_{q,a}(\mathbf{Z})$, and $\mathcal{L}_a$ be as defined in Theorem 7.6. We set $L_a = \#\mathcal{L}_a$. Then, we can fix $t \times L_a$ variables $\mathbf{Z}_i$ to arbitrary values in $\mathcal{W}_{q,a}(\mathbf{Z})$. The system obtained has m solutions (counted without multiplicities), one for each sum $\sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^{\ell q^e}, \mathbf{y}^{\ell q^e})$ with $0 \leqslant e \leqslant m-1$.*

Our purpose is to find a basis of one of the vector spaces $\sum_{\ell\in\mathcal{L}_a}\mathrm{GRS}_t(\mathbf{x}^{\ell q^e},\mathbf{y}^{\ell q^e})$. To do so, we pick L_at independent solutions of $\mathcal{W}_{q,a}(\mathbf{Z})$ by fixing the variables $Z_0,Z_1,\ldots,Z_{L_at-1}$ in $\mathcal{W}_{q,a,t}(\mathbf{Z})$ accordingly. Namely, for $0\leqslant i\leqslant L_at-1$, we pick one solution $\mathbf{v}^{(i)}$ among the m solutions of the system

$$\mathcal{W}_{q,a}(\mathbf{Z})\bigcup\{Z_i=1,Z_j=0\mid 0\leqslant j\neq i\leqslant L_at-1\}.$$

According to Definition 1.17, we know those solutions can be written as follows,

$$\mathbf{v}^{(i)}=\left(0,\ldots,1,\ldots,0,\sum_{\ell\in\mathcal{L}_a}y_{L_at}^{q_i}Q_{i,\ell}(x_{L_at}^{q_i}),\ldots,\sum_{\ell\in\mathcal{L}_a}y_{n-1}^{q_i}Q_{i,\ell}(x_{n-1}^{q_i})\right)\in\sum_{\ell\in\mathcal{L}_a}\mathrm{GRS}_t(\mathbf{x}^{\ell q^{e_i}},\mathbf{y}^{\ell q^{e_i}}),\tag{7.6}$$

for some power of the characteristic $q_i=q^{e_i},0\leqslant e_i\leqslant m-1$ and $Q_{i,\ell}\in\mathbb{F}_{q^m}[z]$ of degree lower than t. After L_at resolutions of $\mathcal{W}_{q,a}(\mathbf{Z})$, the solutions $\mathbf{v}^{(i)}$ are not necessarily a basis of one of the vector spaces $\sum_{\ell\in\mathcal{L}_a}\mathrm{GRS}_t(\mathbf{x}^{\ell q^e},\mathbf{y}^{\ell q^e})$ because the Fröbenius exponents need not be identical for all $\mathbf{v}^{(i)}$'s. In practice, this is not an issue, as explained below.

Simplication: Fröbenius Alignment. Let $\{\mathbf{v}^{(i)}\}_{0\leqslant i\leqslant L_at-1}$ be as defined in (7.6). We can suppose without loss of generality that $q_0=q_1=\ldots=q_{L_at-1}$. This simplification requires less than $m^{(L_at-1)}$ Fröbenius evaluations on the solutions. Indeed, $\mathbf{v}\in\sum_{\ell\in\mathcal{L}_a}\mathrm{GRS}_t(\mathbf{x}^{\ell q^e},\mathbf{y}^{\ell q^e})$, implies that $\mathbf{v}^q\in\sum_{\ell\in\mathcal{L}_a}\mathrm{GRS}_t(\mathbf{x}^{\ell q^{e+1}},\mathbf{y}^{\ell q^{e+1}})$. For the parameters considered in [BLP11b, BLP11c], m and t are rather small, making the cost of the Fröbenius alignment negligible. In the rest of this article, we assume that $q_0=\ldots=q_{L_at-1}=1$.

Example 7.9. *Pick for instance $q=8$ and solve the system $\mathcal{W}_{q,a}$ with $a=2$. Thanks to Theorem 7.6, after re-alignment of the Fröbenius exponents, we have a basis of the vector space* $\mathrm{GRS}_t(\mathbf{x},\mathbf{y})+\mathrm{GRS}_t(\mathbf{x}^2,\mathbf{y}^2)$*, that is:*

$$\left\{(y_iQ(x_i)+y_i^2R(x_i^2))_{0\leqslant i\leqslant n-1}|Q,R\in\mathbb{F}_{q^m}[z],\deg(Q),\deg(R)\leqslant t-1\right\}.$$

Once we know a basis $(\mathbf{v}^{(i)})_{0\leqslant i\leqslant L_at-1}$ of $\sum_{\ell\in\mathcal{L}_a}\mathrm{GRS}_t(\mathbf{x}^\ell,\mathbf{y}^\ell)$, we aim at recovering the basis of a single GRS code. This is what we call the *disentanglement*.

7.2.2 Disentanglement of the System Solutions

The Sidelnikov-Shestakov [SS92] attack – that we briefly recalled in Paragraph 2.3.1 – is a well known attack against McEliece schemes instantiated with GRS codes [Nie86]. In our case, we can have a sum of GRS codes. In this situation, it seems not possible to apply directly [SS92], as the vectors of $\sum_{\ell\in\mathcal{L}_a}\mathrm{GRS}_t(\mathbf{x}^\ell,\mathbf{y}^\ell)$ do not have the desired form, that is $(y_0Q(x_0),\ldots,y_{n-1}Q(x_{n-1}))$. To overcome this issue, we propose to use well-chosen intersections to recover a basis suitable for Sidelnikov-Shestakov. To gain intuition, we provide a small example.

Example 7.10. *We continue with the example 7.9. By squaring all the elements of* $\mathrm{GRS}_t(\mathbf{x},\mathbf{y})+\mathrm{GRS}_t(\mathbf{x}^2,\mathbf{y}^2)$*, we have a basis of* $\mathrm{GRS}_t(\mathbf{x}^2,\mathbf{y}^2)+\mathrm{GRS}_t(\mathbf{x}^4,\mathbf{y}^4)$*:*

$$\left\{(y_i^2Q(x_i^2)+y_i^4R(x_i^4))_{0\leqslant i\leqslant n-1}|Q,R\in\mathbb{F}_{q^m}[z],\deg(Q),\deg(R)\leqslant t-1\right\}.$$

We prove in Proposition 7.12 that, in charac. 2*,*

$$\big(\mathrm{GRS}_t(\mathbf{x},\mathbf{y})+\mathrm{GRS}_t(\mathbf{x}^2,\mathbf{y}^2)\big)\cap\big(\mathrm{GRS}_t(\mathbf{x}^2,\mathbf{y}^2)+\mathrm{GRS}_t(\mathbf{x}^4,\mathbf{y}^4)\big)=\mathrm{GRS}_t(\mathbf{x}^2,\mathbf{y}^2).$$

Hence, we have a basis of $\mathrm{GRS}_t(\mathbf{x}^2,\mathbf{y}^2)$*.*

Our general method to disentangle the solutions is proved in characteristic 2, but for other characteristics we need the following assumption:

Assumption 7.11. *Let $q = p^s$ with p prime. Let $\mathbf{x} \in \mathbb{F}_{q^m}^n$ be a support and $\mathbf{y} \in \mathbb{F}_{q^m}^n$ be defined by $\mathbf{y} = g(\mathbf{x})^{-1}$ for some polynomial $g(z) \in \mathbb{F}_{q^m}[z]$ of degree t. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two subsets of $\{1, \dots, q\}$ with $(\#\mathcal{L}_1 + \#\mathcal{L}_2)t < n$. Then, we have that:*

$$\left(\sum_{\ell \in \mathcal{L}_1} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)\right) \bigcap \left(\sum_{\ell \in \mathcal{L}_2} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)\right) = \sum_{\ell \in \mathcal{L}_1 \cap \mathcal{L}_2} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell).$$

For the specific sets $\mathcal{L} \subset \{1, \dots, q\}$ that we encountered, this assumption is rigorously proved in characteristic 2 (see Proposition 7.12). For bigger characteristics, though we could not find a formal proof, we launched more than $100,000$ experiments with parameters taken from the challenges and found out that equality held in all cases.

Now we generalize the method of intersection of codes proposed in Example 7.10.

Proposition 7.12. *Let $q = p^s$ (p prime and $s \geqslant 0$), and $\mathbf{x}, \mathbf{y} \in \mathbb{F}_{q^m}^n$ with $\mathbf{y}^{-1} = g(\mathbf{x})$ where $g(z) \in \mathbb{F}_{q^m}[z]$ of degree $t > 0$. Let also $a, 0 < a < s$, and*

$$\mathcal{L}_a = \bigcup_{0 \leqslant r \leqslant s-1-a} \{p^r, 2p^r, \dots, (p-1)p^r\} \cup \left\{p^{s-a}\right\}.$$

Then, under the following hypotheses:
— if $p = 2$, we suppose that $g(z)$ is irreducible and $(t-1)p^{2(s-a)} < n$,
— if $p > 2$, we suppose that Assumption 7.11 is verified,
it holds that:

$$\sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell) \cap \left(\sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)\right)^{p^{(s-a)}} = \mathrm{GRS}_t(\mathbf{x}^{p^{s-a}}, \mathbf{y}^{p^{s-a}}).$$

We separate the proof according to $p = 2$ or $p > 2$.

Proof relying on Assumption 7.11 for $p > 2$.

Proof. Let $\Phi : (m_0, \dots, m_{n-1}) \in \mathbb{F}_{q^m}^n \mapsto (m_0^{p^{s-a}}, \dots, m_{n-1}^{p^{s-a}})$. First, remark that, as p^{s-a} is a power of the characteristic, it holds that $\Phi\left(\mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)\right) = \mathrm{GRS}_t(\mathbf{x}^{p^{s-a}\ell}, \mathbf{y}^{p^{s-a}\ell})$ for all ℓ, and $\Phi\left(\sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)\right) = \sum_{\ell \in \mathcal{L}'_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)$ with $\mathcal{L}'_a = \bigcup_{s-a \leqslant r \leqslant 2(s-a)-1} \{p^r, 2p^r, \dots, (p-1)p^r\} \cup \left\{p^{2(s-a)}\right\}$. When $p = 2$, we fully prove the proposition just below. Otherwise (when $p > 2$), we rely on Assumption 7.11 with the sets $\mathcal{L}_a$ and $\mathcal{L}'_a$. Then, we have $\mathcal{L}_a \cap \mathcal{L}'_a = \{p^{s-a}\}$, and the desired equality. □

Complete proof when $p = 2$.

Proof. When $p = 2$, we prove Proposition 7.12 without resorting to Assumption 7.11. We use the fact that the polynomial $g(z)$ linking $\mathbf{x}$ and $\mathbf{y}^{-1}$ is irreducible. This is the case in the construction proposed in [BLP11b, BLP11c]. For $p = 2$, $\mathcal{L}_a$ is reduced to powers of 2, namely $\mathcal{L}_a = \{2^u\}_{0 \leqslant u \leqslant s-a}$. In such case, it holds that

$$\sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell) \bigcap \Phi\left(\sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell)\right) = \left(\sum_{u=0}^{s-a} \mathrm{GRS}_t(\mathbf{x}^{2^u}, \mathbf{y}^{2^u})\right) \cap \left(\sum_{u=s-a}^{2(s-a)} \mathrm{GRS}_t(\mathbf{x}^{2^u}, \mathbf{y}^{2^u})\right).$$

So the proof consists in showing that this intersection is reduced to $\mathrm{GRS}_t(\mathbf{x}^{2^{s-a}}, \mathbf{y}^{2^{s-a}})$. We pick $\mathbf{v} \in \left(\sum_{u=0}^{s-a} \mathrm{GRS}_t(\mathbf{x}^{2^u}, \mathbf{y}^{2^u})\right) \cap \left(\sum_{u=s-a}^{2(s-a)} \mathrm{GRS}_t(\mathbf{x}^{2^u}, \mathbf{y}^{2^u})\right)$. There exist polynomials $R_{2^u}, Q_{2^{s-a+u}} \in \mathbb{F}_{q^m}[z]$ (with $0 \leqslant u \leqslant s-a$) of degree lower than t such that

$$v_i = \sum_{u=0}^{s-a} y_i^{2^u} R_{2^u}(x_i^{2^u}) = \sum_{u=0}^{s-a} y_i^{2^{s-a+u}\ell} Q_{2^{s-a+u}}(x_i^{2^{s-a+u}\ell}), \text{ for all } 0 \leqslant i \leqslant n-1,$$

As $y_i = g(x_i)^{-1}$, we obtain polynomial relations in the x_i's by multiplying by $g(x_i)^{2^{2(s-a)}}$. This yields n relations:

$$\sum_{u=0}^{s-a} g(x_i)^{(2^{2(s-a)-u})} R_{2^u}(x_i^{2^u}) = \sum_{u=0}^{s-a} g(x_i)^{2^{2(s-a)-(s-a+u)}} Q_{2^{s-a+u}}(x_i^{2^{s-a+u}}).$$

We suppose here that the degree of this polynomial relation is lower than n, that is $(t-1)2^{2(s-a)} < n$, so that we can deduce the polynomial equality:

$$\sum_{u=0}^{s-a} g(z)^{(2^{2(s-a)-u})} R_{2^u}(z^{2^u}) = \sum_{u=0}^{s-a} g(z)^{2^{2(s-a)-(s-a+u)}} Q_{2^{s-a+u}}(z^{2^{s-a+u}}). \tag{7.7}$$

Modulo $g(z)$ all polynomials vanish but one, this yields $Q_{2^{2(s-a)}}(z^{2^{2(s-a)}}) \equiv 0 \mod g(z)$. Thanks to Lemma 7.7, we have $g(z)$ divides $Q_{2^{2(s-a)}}(z^{2^{2(s-a)}}) = \left(Q^{(u)}_{2^{2(s-a)}}(z)\right)^{2^{2(s-a)}}$ (for $u = ms - 2(s-a)$). As $g(z)$ is irreducible, this entails that $g(z)$ divides $Q^{(u)}_{2^{2(s-a)}}(z)$, but $Q^{(u)}_{2^{2(s-a)}}(z)$ has same degree as $Q_{2^{2(s-a)}}(z)$, which has degree lower than t (notations as in the proof of Theorem 7.6). Hence we deduce that $Q^{(u)}_{2^{2(s-a)}}(z) = 0$ and also its Fröbenius $Q_{2^{2(s-a)}} = 0$. Then, we look at the new relation of type (7.7) and start over with the polynomial $Q_{2^{2(s-a)-1}}(z^{2^{2(s-a)-1}})$. The proof that $Q_{2^{2(s-a)-1}} = 0$ is identical. One after the other, we prove that all the polynomials $R_{2^u}, Q_{2^{s-a+u}}$ are zero except the matching polynomials $R_{2^{s-a}}$ and $Q_{2^{s-a}}$ which are equal, so that $\mathbf{z} \in \mathrm{GRS}_t(\mathbf{x}^{2^{s-a}}, \mathbf{y}^{2^{s-a}})$. The problem when $p \neq 2$ is that the set $\mathcal{L}_a$ contains exponents which are not a pure power of p. □

7.2.3 Sidelnikov-Shestakov Adapted To Recover the Goppa Polynomial

Once a basis of $\mathrm{GRS}_t(\mathbf{x}^{p^{s-a}}, \mathbf{y}^{p^{s-a}})$ is known, we recover $\mathbf{x}, \mathbf{y}$ and $g(z)$ thanks to a variant of Sidelnikov-Shestakov. The specificity of the GRS codes that we attack lies in the fact that there exists a polynomial relation between the support and the multipliers.

Proposition 7.13. *Let $\mathbf{x}$ be an n-tuple $(x_0, \ldots, x_{n-1})$ of distinct elements of $\mathbb{F}_{q^m}$ and $g(z) \in \mathbb{F}_{q^m}[z]$ be a squarefree polynomial of degree t such that $g(x_i) \neq 0$, for all $i, 0 \leqslant i \leqslant n-1$ Let $\mathbf{G}_{\mathrm{GRS}}$ be the generator matrix of a GRS code $\mathrm{GRS}_t(\mathbf{x}, \Gamma(\mathbf{x})^{-1})$. There is a polynomial-time algorithm which allows to recover a n-tuple $\mathbf{x}' = (x'_0, \ldots, x'_{n-1})$ of distinct elements of $\mathbb{F}_{q^m}$ and a squarefree polynomial $\Gamma'(z) \in \mathbb{F}_{q^m}[z]$ of degree t such that $g'(x'_i) \neq 0$, for all $i, 0 \leqslant i \leqslant n-1$ and $\mathrm{GRS}_t(\mathbf{x}, \Gamma(\mathbf{x})^{-1}) = \mathrm{GRS}_t(\mathbf{x}', \Gamma'(\mathbf{x}')^{-1})$.*

This problem is very close to the one addressed in [SS92]. The only issue is that the homographic transformation on the support used in the original attack (see Paragraph 2.3.1) indeed preserves the GRS structure but not the polynomial link. Thus, polynomial interpolation over $\mathbf{x}$ and $\mathbf{y}^{-1}$ is not possible. We propose to avoid this homographic transformation by considering a well chosen extended code (Definition 1.9).

Our algorithm, proved below, is the following.

Algorithm 8 Extended Version of Sidelnikov-Shestakov algorithm

INPUT : $\mathbf{G}_{GRS}$ a generator matrix of $\mathscr{C}_{GRS} = \mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$, with $\mathbf{y} = \Gamma(\mathbf{x})^{-1}$ ($\deg(\Gamma) = t$)
OUTPUT : Secret $\mathbf{x}$, $\mathbf{y}$, and $\Gamma(z)$

1: Build $\mathbf{P} = (p_{i,j})_{\substack{0 \leqslant i \leqslant n-t-1 \\ 0 \leqslant j \leqslant n-1}}$ a generator matrix of the dual of $\mathscr{C}_{GRS}$.
2: Deduce $\tilde{\mathbf{P}}$ a matrix of the extended code (Definition 1.9) of the code spanned by $\mathbf{P}$.
3: Build $\big(\mathbf{I}_t | \mathbf{U}\big)$, with $\mathbf{U} = (u_{i,j})_{\substack{0 \leqslant i \leqslant t \\ t+1 \leqslant j \leqslant n}}$ a parity-check matrix of the code spanned by $\tilde{\mathbf{P}}$ in systematic form.
4: Solve the linear system with unknowns X_i's to find $\mathbf{x}$

$$\left\{ \frac{u_{i,j}}{u_{i',j}}(X_{i'} - X_j) = \frac{u_{i,n}}{u_{i',n}}(X_i - X_j) \mid 0 \leqslant i, i' \leqslant t, t+1 \leqslant j \leqslant n-1 \right\}.$$

5: Solve the linear system with unknowns Y_i's to find $\mathbf{y}$ (the x_i's were found at previous step)

$$\left\{ \sum_{i=0}^{n-1} p_{j,i} x_i^{\ell} Y_i = 0 \mid 0 \leqslant j \leqslant n-t-1, 0 \leqslant \ell \leqslant t-1 \right\}.$$

6: Interpolate $\Gamma(z)$ from $\mathbf{x}$ and $\mathbf{y}^{-1}$.

Proposition 7.14. *With input* $\mathbf{G}$ *the generator matrix of a* $[n,t]_q$ GRS *code* $\mathscr{C}$*, Algorithm 8 outputs vectors* $\mathbf{x}, \mathbf{y} \in \mathbb{F}_{q^m}^n$ *and a polynomial* $\Gamma \in \mathbb{F}_{q^m}[z]$ *of degree* t *such that* $\mathscr{C} = \mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$ *and* $\mathbf{y} = \Gamma(\mathbf{x})^{-1}$.

Proof. From a generator matrix of $\mathrm{GRS}_t(\mathbf{x}, \mathbf{y})$, we deduce easily (by linear algebra) a parity-check matrix $\widetilde{\mathbf{P}}$ of $\widetilde{\mathscr{C}^{\perp}}$ in sytematic form. As the dual of $\mathscr{C}_{GRS}$ is the Goppa code $\mathscr{G}(\mathbf{x}, g)$ over $\mathbb{F}_{q^m}$, we know thanks to Proposition 1.24 that $\widetilde{\mathbf{V}}_{t+1}(\mathbf{x}, \mathbf{y})$ is parity check matrix of $\widetilde{\mathscr{C}_{GRS}^{\perp}}$, with

$$\widetilde{\mathbf{V}}_{t+1}(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} y_0 & \cdots & y_{n-1} & 0 \\ y_0 x_0 & \cdots & y_{n-1} x_{n-1} & 0 \\ \vdots & & \vdots & \vdots \\ y_0 x_0^{t-1} & \cdots & y_{n-1} x_{n-1}^{t-1} & 0 \\ y_0 x_0^{t} & \cdots & y_{n-1} x_{n-1}^{t} & \frac{1}{\mathrm{LC}(g)} \end{pmatrix}$$

So, the rows of $\widetilde{\mathbf{P}}$ are linear combinations of the rows of $\widetilde{\mathbf{V}_t}(\mathbf{x}, \mathbf{y})$, so we write it:

$$\begin{aligned} \widetilde{\mathbf{P}} &= \begin{pmatrix} 1 & 0 & \dots & 0 & y_{t+1} R_0(x_{t+1}) & \dots & y_{n-1} R_0(x_{n-1}) & \frac{\mathrm{LC}(R_0)}{\mathrm{LC}(g)} \\ 0 & 0 & \ddots & 0 & & \vdots & & \\ 0 & 0 & \dots & 1 & y_{t+1} R_t(x_{t+1}) & \dots & y_{n-1} R_t(x_{n-1}) & \frac{\mathrm{LC}(R_t)}{\mathrm{LC}(g)} \end{pmatrix} \\ &= \left(\begin{array}{c|ccc} & u_{0,t+1} & \dots & u_{t+1,n} \\ \mathbf{I}_t & & \vdots & \\ & u_{t,t+1} & \dots & u_{t+1,n} \end{array} \right) \end{aligned}$$

where $R_0, \dots, R_t$ are polynomials of $\mathbb{F}_{q^m}[z]$ of degree $t+1$.

Now, we perform the Sidelnikov-Shestakov like method $\widetilde{\mathbf{P}}$. Since we know t roots of each R_j (of degree t), we have that

$$R_j(z) = \mathrm{LC}(R_j) \prod_{0 \leqslant i \leqslant t, i \neq j} (z - x_i).$$

This allows to write for $0 \leqslant i, i' \leqslant t$ and $t+1 \leqslant j \leqslant n-1$,

$$\frac{u_{i,j}}{u_{i',j}} = \frac{\mathrm{LC}(R_i)(x_i - x_j)}{\mathrm{LC}(R_{i'})(x_{i'} - x_j)}, \text{ and for } j = n, \ \frac{u_{i,n}}{u_{i',n}} = \frac{\mathrm{LC}(R_i)}{\mathrm{LC}(R_{i'})}.$$

Note that none of the coefficients $u_{i,j}$'s can be zero, as it would provide a $t+1$-th root for a Q_i. Finally, the support elements are solutions of the linear system:

$$\left\{ \frac{u_{i,j}}{u_{i',j}}(x_{i'} - x_j) = \frac{u_{i,n}}{u_{i',n}}(x_i - x_j) \mid 0 \leqslant i, i' \leqslant t, t+1 \leqslant j \leqslant n-1 \right\}.$$

□

7.2.4 Recovery of the Incognito Polynomial by Solving a Linear System

Until now, we recovered a secret support and the polynomial $g(z)$. We saw that the full Goppa polynomial can contain other factors, for instance the factor $f(z)$ in Wild McEliece Incognito.

To recover this factor, we find the multipliers associated to f, that is the vector $\mathbf{w} = f(\mathbf{x})^{-1}$. Then, we perform polynomial interpolation. We note that once $\mathbf{x}$ and $\mathbf{y} = g(\mathbf{x})^{-1}$ are known, then many of the equations of Lemma 7.3 become linear in $\mathbf{w}$. Namely,

$$\bigcup_{u_y=1}^{q} \left\{ \sum_{j=0}^{n-1} g_{i,j} w_j \left(y_j^{u_y} x_j^{u_x} \right) = 0 \quad | \quad 0 \leqslant i \leqslant k-1, 0 \leqslant u_x \leqslant u_y t + \deg(f) - 1 \right\}.$$

In practice, we observed that the linear system obtained has a rank defect and is not sufficient to find $\mathbf{w}$. However, we can also use the fact that $\mathscr{C}_{pub} \subset \mathscr{G}(\mathbf{x}, f(z))$ and Proposition 1.24 to prove that

$$\sum_{j=0}^{n-1} g_{i,j} w_i x_i^{\deg(f)} = \frac{1}{\mathrm{LC}(f)} \left(\sum_{j=0}^{n-1} g_{i,j} \right).$$

Since $\mathbf{x}$ is known and setting $\mathrm{LC}(f) = 1$, we obtain new linear equations in the components of $\mathbf{w}$. Putting all the linear equations together, experiments show then that we obtain a unique solution $\mathbf{w}$, and f by polynomial interpolation.

To sum up, the results of this section prove the following theorem.

Theorem 7.15. *Let $q = p^s$ (p prime and $s \geqslant 0$). Let $\mathbf{G}_{pub} = (g_{i,j})_{\substack{0 \leqslant i \leqslant k-1 \\ 0 \leqslant j \leqslant n-1}}$ be a generator matrix of a masked Wild Goppa code $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}, f(z)g^{q-1}(z))$. Let $\mathbf{y} = g(\mathbf{x})^{-1}$, $t = \deg(g)$, and*

$$\mathcal{V} = \left(\sum_{\ell \in \mathcal{L}_a} \mathrm{GRS}_t(\mathbf{x}^\ell, \mathbf{y}^\ell) \right) \text{ where } \mathcal{L}_a = \bigcup_{0 \leqslant r \leqslant s-1-a} \{p^r, 2p^r, \ldots, (p-1)p^r\} \cup \{p^{s-a}\}.$$

Once $\mathcal{V}$ is given, we can recover in polynomial-time a support $\mathbf{x}'$ and polynomials $f'(z), g'(z) \in \mathbb{F}_{q^m}[z]$ such that $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}', f'g'^{q-1})$. Stated differently, we can recover in polynomial-time a key $(\mathbf{x}', g', f')$ equivalent to the secret-key as soon as the system (7.5) has been solved.

7.3 Application to Wild McEliece & Incognito

We applied the framework of our attack to Wild McEliece & Incognito keys proposed by the authors of [BLP11b, BLP11c]. While the attack method we presented so far can be applied as soon as the Goppa polynomials contains a factor with multiplicity greater than 1 (it does not have to be equal to q), we found out that a multiplicity equal to the the size of the base field entailed a very specific property when the base field is not a prime field. Namely, from the public matrix, we can construct a new code with same private elements. We explain this in Paragraph 7.3.1. Then, we apply all the previous results in Paragraph 7.3.2 to evaluate the security of the keys proposed in [BLP11b, BLP11c].

7.3.1 A property of Wild Goppa codes over non-prime base fields

The most (computationally) difficult part of our attack against Wild McEliece & Incognito is to solve the algebraic system defined in Theorem 7.6. In this part, we aim at giving a better idea of the complexity of resolution by determining the exact number of "free variables" in the system. Namely, we show that we can eliminate many variables thanks to linear equations. The system $\mathcal{W}_{q,a}(\mathbf{Z}) = \bigcup_{u\in\mathcal{P}_a}\left\{\sum_{j=0}^{n-1} g_{i,j}Z_j^u = 0 \quad | \quad 0 \leqslant i \leqslant k-1\right\}$ of Theorem 7.6 obviously contains k linear equations by picking $u = 1$ ($1 \in \mathcal{P}_a$ by definition). We can easily derive other linear equations by applying the additive map $z \mapsto z^{(q^m/p^u)}$ to all the equations in degree p^u. As the solutions lie in $\mathbb{F}_{q^m}$, it holds that $(Z_j^{p^u})^{q^m/p^u} = Z_j$, and for $0 \leqslant i \leqslant k-1$

$$\left(\sum_{j=0}^{n-1} g_{i,j}Z_j^{p^u}\right)^{q^m/p^u} = \sum_{j=0}^{n-1} g_{i,j}^{q^m/p^u} Z_j = 0.$$

However, we observed that those linear equations were very redundant. To explain those linear dependencies, we found out a property of the masked Wild Goppa codes $\mathscr{G}(\mathbf{x}, f(z)g(z)^q)$ (Theorem 7.16). We prove that, by simple operations on their generator matrices, we can build a generator matrix of the code $\mathscr{G}(\mathbf{x}, g(z)^p)$ over $\mathbb{F}_p$. This latter matrix allows to write many *independent* linear equations implying the private elements of $\mathscr{C}_{pub}$.

Namely, we consider the scalar restriction of $\mathbf{m} \in \mathscr{C}_{pub} \subseteq \mathbb{F}_q^n$ into $\mathbb{F}_p^s$. This yields s components $\mathbf{m}^{(0)}, \ldots, \mathbf{m}^{(s-1)} \in \mathbb{F}_p^n$ (we write each $\mathbf{m} \in \mathbb{F}_q^n$ over a $\mathbb{F}_p$-basis, i.e. $\mathbf{m} = \mathbf{m}^{(0)}\theta_0 + \cdots + \mathbf{m}^{(s-1)}\theta_{s-1}$). Let $\mathscr{C}^{\mathbb{F}_p} \subseteq \mathbb{F}_p^n$ be the code generated by the coordinate vectors $\mathbf{m}^{(0)}, \ldots, \mathbf{m}^{(s-1)}$ for all the codewords $\mathbf{m} \in \mathscr{C}_{pub}$. We explain how to build a generator matrix of $\mathscr{C}^{\mathbb{F}_p}$.

For a vector $\mathbf{m} \in (\mathbb{F}_q)^n$, the associated coordinate vectors are obtained by linear combinations of $\mathbf{m}$ and its Fröbenius $\mathbf{m}^p, \ldots, \mathbf{m}^{p^{s-1}}$. In details, for an element $m \in \mathbb{F}_q$, we write over an $\mathbb{F}_p$-basis $m = m_0\theta_0 + \cdots + m_{s-1}\theta_{s-1}$. As $m_i \in \mathbb{F}_p$, we have the relations $m^{p^j} = m_0\theta_0^{p^j} + \cdots + m_{s-1}\theta_{s-1}^{p^j}$ that we write matricially:

$$\begin{pmatrix} m \\ m^p \\ \vdots \\ m^{p^{s-1}} \end{pmatrix} = \underbrace{\begin{pmatrix} \theta_0 & \ldots & \theta_{s-1} \\ \theta_0^p & \ldots & \theta_{s-1}^p \\ & \vdots & \\ \theta_0^{p^{s-1}} & \ldots & \theta_{s-1}^{p^{s-1}} \end{pmatrix}}_{\mathbf{C}} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{s-1} \end{pmatrix}.$$

$\mathbf{C}$ is invertible since the θ_i's are a basis [LN97, p.62 Corollary 28]. $\mathbf{C}^{-1}$ allows to embed a

packet of rows $\begin{pmatrix} \mathbf{m} \\ \mathbf{m}^p \\ \vdots \\ \mathbf{m}^{p^{s-1}} \end{pmatrix}$ into $(\mathbb{F}_p)^n$. The product $\mathbf{C}^{-1} \begin{pmatrix} \mathbf{m} \\ \mathbf{m}^p \\ \vdots \\ \mathbf{m}^{p^{s-1}} \end{pmatrix}$ gives exactly the s rows of the coordinates of $\mathbf{m}$ over $\mathbb{F}_p$. To write a matrix of $\mathscr{C}^{(\mathbb{F}_p)}$, expand a generator matrix $\mathbf{G}_{pub}$ of $\mathscr{C}_{pub}$ as follows: suppose $\mathbf{G}_{pub}$ has rows $L_0, \ldots, L_{k-1}$, build $\mathbf{G}^{(F)}$ the matrix with rows $L_0, L_0^p, \ldots, L_0^{p^{s-1}}, L_1, L_1^p, \ldots$. The matrix

$$\widetilde{\mathbf{C}^{-1}} = \left(\begin{array}{c|c|c} \mathbf{C}^{-1} & \ldots & 0_s \\ \hline & \ddots & \\ \hline 0_s & \ldots & \mathbf{C}^{-1} \end{array} \right),$$

is such that $\widetilde{\mathbf{C}^{-1}}\mathbf{G}^{(F)}$ has coefficients in $\mathbb{F}_p$ and is a generator matrix of $\mathscr{C}^{\mathbb{F}_p}$.

Theorem 7.16. *Let $q = p^s$ (p prime, and $s > 0$). Let $\mathbf{G}_{pub} = (g_{i,j})_{\substack{0 \leqslant i \leqslant k-1 \\ 0 \leqslant j \leqslant n-1}}$ be a generator matrix of a masked Wild Goppa code $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}, f(z)g(z)^{q-1})$. Then, the code $\mathscr{C}^{\mathbb{F}_p}$ over $\mathbb{F}_p$ spanned by $\widetilde{\mathbf{C}^{-1}}\mathbf{G}^{(F)}$ (as introduced previously) satisfies:*

$$\mathscr{C}^{\mathbb{F}_p} \subseteq \mathscr{G}(\mathbf{x}, g(z)^p)_{|\mathbb{F}_p}.$$

Proof. Recall that $q = p^s$ (p prime, and $s > 0$ here). We find a code common to the vectors $\mathbf{m}, \ldots, \mathbf{m}^{p^{s-1}}$ for $\mathbf{m} \in \mathscr{G}(\mathbf{x}, g^q(z))$.

Proposition 7.17. *Let $\mathscr{C}^{(F)}$ be the code generated by the rows of $\mathscr{C}_{pub}$ and their Frobenius. Then it holds that*

$$\mathscr{C}^{(F)} = \left\{ \mathbf{m}, \mathbf{m}^p, \ldots, \mathbf{m}^{p^{s-1}} \mid \mathbf{m} \in \mathscr{C}_{pub} \right\} \subset \mathcal{A}_{pt}(\mathbf{x}^{p^{s-1}}, \mathbf{y}^{p^s}).$$

Proof. First, we show that $\mathbf{m}^{p^j} \in \mathcal{A}_{p^{s-j}}(\mathbf{x}^{p^j}, \mathbf{y}^{p^s})$. As we know that $\mathbf{m} \in \mathscr{G}(\mathbf{x}, g(z)^q)$, it holds that

$$g(z)^q \left| \sum_{i=0}^{n-1} \frac{m_i}{z - x_i} \right. .$$

We apply the j-th power of the Fröbenius, to all the coefficients of the fractions appearing in the previous relation:

$$(g^{(j)}(z))^q \left| \sum_{i=0}^{n-1} \frac{m_i^{p^j}}{z - x_i^{p^j}} \right.$$

which means exactly that $\mathbf{m}^{p^j} \in \mathscr{G}(\mathbf{x}^{p^j}, (g^q)^{(j)})(z))$. Remark that $\left(g^{(j)}\right)^q = (g^q)^{(j)}$. We know from the proof of Proposition 7.3 that for any $P \in \mathbb{F}_{q^m}[z]$, $\mathscr{G}(\mathbf{x}^{p^j}, P^q(z))$ is included in any $\mathscr{G}(\mathbf{x}^{p^j}, P^\ell(z))$ with $1 \leqslant \ell \leqslant q = p^s$. In particular, with $\ell = p^{s-j}$ and $P = g^{(j)}$, it holds that

$$\mathbf{m}^{p^j} \in \mathscr{G}(\mathbf{x}^{p^j}, (g^{(j)}(z))^{p^{s-j}}) = \mathcal{A}_{p^{s-j}t}(\mathbf{x}^{p^j}, \mathbf{y}^{p^s}).$$

The latter equality can be detailed as follows. According to Lemma 7.7, $(g^{(j)}(z))^{p^{s-j}} = g^{(s)}(z^{p^{s-j}})$, so the multipliers associated to $\mathscr{G}(\mathbf{x}^{p^j}, (g^{(j)}(z))^{p^{s-j}})$ are

$$\left(g^{(s)}(x_i^{p^j p^{s-j}})\right)^{-1} = \left(g(x_i)^{p^s}\right)^{-1} = y_i^{p^s}.$$

Looking at the parity-check matrices of the codes $\mathcal{A}_{p^{s-j}t}(\mathbf{x}^{p^j}, \mathbf{y}^{p^s})$, we see that:

$$\mathscr{C}_{pub} = \mathcal{A}_{p^s t}(\mathbf{x}, \mathbf{y}^{p^s}) \subset \mathcal{A}_{p^{s-1}t}(\mathbf{x}^p, \mathbf{y}^{p^s}) \subset \cdots \subset \mathcal{A}_t(\mathbf{x}^{p^s}, \mathbf{y}^{p^s}).$$

Thus, for $\mathbf{m} \in \mathscr{C}_{pub}$, we have $\mathbf{m}, \mathbf{m}^p, \ldots, \mathbf{m}^{p^{s-1}} \in \mathcal{A}_{pt}(\mathbf{x}^{p^{s-1}}, \mathbf{y}^{p^s})$.

□

So far, we know that

$$\mathscr{C}^{(\mathbb{F}_p)} \subset \mathcal{A}_{pt}(\mathbf{x}^{p^{s-1}}, \mathbf{y}^{p^s}).$$

This implies that $\mathbf{G}^{(F)}\mathbf{V}_{pt}(\mathbf{x}^{p^{s-1}}, \mathbf{y}^{p^s}) = \mathbf{0}$. Thanks to $\widetilde{\mathbf{C}^{-1}}$, we have:

$$\underbrace{\widetilde{\mathbf{C}^{-1}}\mathbf{G}^{(F)}}_{\in \mathcal{M}(\mathbb{F}_p)} \times \mathbf{V}_{pt}(\mathbf{x}^{p^{s-1}}, \mathbf{y}^{p^s}) = \mathbf{0}.$$

We can apply the Fröbenius to all the equations in the matrix product, or rather extract a p^{s-1} root, it does not change any coefficient in $\widetilde{\mathbf{C}^{-1}}\mathbf{G}^{(F)}$, and it yields:

$$\widetilde{\mathbf{C}^{-1}}\mathbf{G}^{(F)} \times \mathbf{V}_{pt}(\mathbf{x}, \mathbf{y}^p) = \mathbf{0}.$$

This means exactly that the rows of $\widetilde{\mathbf{C}^{-1}}\mathbf{G}^{(F)}$ generate a subcode of $\mathscr{G}(\mathbf{x}, g(z)^p)$, which is a Wild Goppa code over $\mathbb{F}_p$. □

In practice, we observed equality in the inclusion provided $s\dim(\mathscr{C}_{pub}) > \dim(\mathscr{G}(\mathbf{x}, g(z)^p))$. Note that $\mathscr{G}(\mathbf{x}, g(z)^p)$ is a Wild Goppa code with the same private elements $\mathbf{x}$ and $\mathbf{y} = g(\mathbf{x})^{-1}$ as $\mathscr{C}_{pub}$. This provides extra equations on the variables $\mathbf{Z}$ of $\mathcal{W}_{q,a}(\mathbf{Z})$.

Proposition 7.18. *Let $\mathscr{C}_{pub} = \mathscr{G}(\mathbf{x}, f(z)g^{q-1}(z))$ and $\mathcal{W}_{q,a}(\mathbf{Z})$ the associated system for $1 \leqslant a \leqslant s$. Let $\tilde{\mathbf{G}}_{\mathbb{F}_p} = (\widetilde{g}_{i,j})_{\substack{0\leqslant i\leqslant k_p-1 \\ 0\leqslant j\leqslant n-1}}$ be a generator matrix of $\mathscr{G}(\mathbf{x}, g(z)^p)$ (with $k_p = \dim(\mathscr{G}(\mathbf{x}, g(z)^p))$). Then, the solutions of $\mathcal{W}_{q,a}(\mathbf{Z})$ satisfy:*

$$\bigcup_{\ell=0}^{p-1} \left\{ \sum_{j=0}^{n-1} \widetilde{g}_{i,j} Z_j = 0 \mid 0 \leqslant u \leqslant t-1, 0 \leqslant i \leqslant k_p - 1 \right\}.$$

Proof. Pick a solution $\mathbf{z}$ with coordinates $z_j = \sum_{\ell \in \mathcal{L}_a} y_j^\ell Q_\ell(x_j^\ell)$, where the Q_ℓ's are polynomials of degree $\leqslant t-1$ of $\mathbb{F}_{q^m}[z]$. We decompose $\mathcal{L}_a = \mathcal{P}_1 \cup \mathcal{P}_2$ with $\mathcal{P}_1 = \{1, 2, \ldots, p^a - 1\}$ and $\mathcal{P}_2 = \{p^a, \ldots, p^s\}$. As $\mathscr{G}(\mathbf{x}, g(z)^p)$ is a Wild Goppa code with the same private elements $\mathbf{x}$ and $\mathbf{y} = g(\mathbf{x})^{-1}$ as $\mathscr{C}_{pub}$, Lemma 7.3 holds for $\ell \in \{1, \ldots, p-1\}$:

$$\sum_{j=0}^{n-1} \widetilde{g}_{i,j} y_j^\ell x_j^{\ell u} = 0 \mid 0 \leqslant u \leqslant t-1, 0 \leqslant i \leqslant k_p - 1$$

Other ℓ's in $\mathcal{L}_a$ can be written $\ell = \ell_0 p^r$ with $\ell_0 \in \{1, \ldots, p-1\}$ and for some $r \leqslant 1$. Apply the additive map $x \mapsto x^{p^r}$ to the relations $\sum_{j=0}^{n-1} \widetilde{g}_{i,j} y_j^{\ell_0} x_j^{\ell_0 u} = 0$. Since $\widetilde{g}_{i,j}^{p^r} = \widetilde{g}_{i,j}$, we obtain $\sum_{j=0}^{n-1} \widetilde{g}_{i,j} y_j^\ell x_j^{\ell u} = 0$. Thus, all the terms $y_j^{u_y} x_j^{u_x}$ in z_j satisfy $\sum_{j=0}^{n-1} \widetilde{g}_{i,j} y_j^{u_y} x_j^{u_x} = 0$, so that $\sum_{j=0}^{n-1} \widetilde{g}_{i,j} z_j = 0$.

□

From Proposition 7.18, we deduce the following corollary.

Corollary 7.19. *As $k_p \geqslant n-(p-1)mst$ (and in practice $k_p = n-(p-1)mst$), the knowledge of $\mathbf{G}_{pub}$ gives access to (at least) $n-(p-1)mst$ independent linear relations between the Z_i's. The system $\mathcal{W}_{q,a}(\mathbf{Z})$ contains (at most) $(p-1)mst$ free variables.*

Remark 7.20. *The number of "free" variables given in Corollary 7.19 is given without taking into account the vector space structure of the solutions. Thanks to Corollary 7.8, we know that $L_a \cdot t$ extra variables can be fixed to arbitrary values in $\mathcal{W}_{q,a}(\mathbf{Z})$.*

For a Goppa polynomial of same degree, but without multiplicities, the number of free variables in the system would be $n-k \geqslant (p^s-1)mt$ instead of $(p-1)mst$. In particular, for a masked code, the number of variables describing it does not depend on the degree of the incognito polynomial f and the attack is not harder for masked codes. This explains why the codes defined over non-prime fields are the weakest ones.

7.3.2 Practical experiments & Broken Challenges

We report below various experimental results performed with our attack on various parameters for which [BLP11b] said that strength is "unclear" and that an attack would not be a "surprise" but for which no actual attack was known. We also generated our own keys/parameters to see how the attack scales. We performed our experiments with off-the-shelf tools (Magma [BCP97b] V2.19-1) and using a 2.93 GHz Intel PC with 128 Gb. of RAM. As the polynomial system solving is by far the most costly step, we give timings only for this one. We performed it using the F_4 algorithm ([Fau99]) of Magma. As explained in Section 7.2, it is necessary to solve the systems $\mathcal{W}_{q,a}(\mathbf{Z})$ a number of times equal to the dimension of the vector space of the solutions (Theorem 7.6). These resolutions are completely independent and can be executed in parallel. This is why we give the timings under the form (number of separate resolutions) $\times$ (time for one resolution). By $\#\mathbf{Z}$, we denote the number of free variables remaining in the system after cleaning up the linear equations (Corollary 7.19) and fixing coordinates thanks to the vector space structure of the solutions (Corollary 7.8). The general formula is $\#\mathbf{Z} = ((p-1)ms - \#\mathcal{L}_a)\,t$ for $q = p^s$ and $s > 1$.

In the experiments, we tried various parameters a for the systems $\mathcal{W}_{q,a}(\mathbf{Z})$. We give a comparison on some examples in Table 7.1 (the system $\mathcal{W}_{q,a}(\mathbf{Z})$ with $a = s$ can be solved in a reasonable amount of time in actually few cases).

q	m	t	n	k	$\deg(f)$	Solving $\mathcal{W}_{q,a}(\mathbf{Z})$ with $a = s$	Solving $\mathcal{W}_{q,a}(\mathbf{Z})$, **optimal** a
32	2	2	678	554	0	2×12s ($\#\mathbf{Z} = 18$)	$8\times$ 0.08 s ($a = 2, \#\mathbf{Z} = 9$)
32	2	1	532	406	32	2×49s ($\#\mathbf{Z} = 9$)	$4\times$ 0.02 s ($a = 2, \#\mathbf{Z} = 6$)
32	2	3	852	621	24	$3\times$ (30 min 46s) ($\#\mathbf{Z} = 37$)	$12\times$ 0.6 s ($a = 2, \#\mathbf{Z} = 18$)
27	3	3	1312	1078	0	$3\times$ (3h 10 min) ($\#\mathbf{Z} = 51$)	$15\times$ 3.0 s ($a = 1, \#\mathbf{Z} = 39$)

Table 7.1 – Comparison of the resolution times of $\mathcal{W}_{q,a}(\mathbf{Z})$ for various possible a's. The smallest possible a gives the best timings.

It appeared that a should be chosen so as to maximize the dimension of the solution set (Theorem 7.6). This choice minimizes the number of variables. Namely, the best choice is to set $a = 1$ when $p > 2$. When $p = 2$, setting $a = 1$ would yield only "linear" equations (of degree $2^u, u \leqslant s$). So, we set $a = 2$ and the systems $\mathcal{W}_{2^u,2}(\mathbf{Z})$ contain only cubic equations. We recall that for $a = s$, Assumption 7.11 is not necessary, whereas we rely on it when $a < s$ and $p \neq 2$. In the rest of the experiments, we always pick the best choice for a.

In Table 7.2, we present experimental results performed with Wild McEliece (when $\deg(f) = 0$) and Incognito ($\deg(f) > 0$) parameters. For Wild McEliece, all the parameters in the scope of our attack were quoted in [BLP11b, Table 7.1] with the international biohazard symbol ☣. The reason is that, for those parameters, enumerating all the possible Goppa polynomials is computationaly feasible. In the current state of the art, to apply the SSA attack ([LS01]), one would not only have to enumerate the irreducible polynomials of $\mathbb{F}_{q^m}[z]$, but also all the possible support sets, as the support-splitting algorithm uses the support set as input (as explained in Paragraph 2.3.2 of Chapter 2).

We indicate, for each set of parameters, the ISD complexity (obtained thanks to Peters' software [1]), as it remains the reference to evaluate the security of a McEliece scheme. We also give the complexity of an SSA attack, which is in the current state-of-the-art $\binom{q^m}{n} \cdot q^{mt}/t$.

Regarding Wild McEliece Incognito, we broke the parameters indicated with a security of 2^{128} in [BLP11c, Table 5.1] for $q \in \{32, 27, 16\}$. For some other non-prime base fields, we give the hardest parameters in the scope of our attack in roughly one day of computation.

For the sake of completeness, we also include in Tables 7.2 Wild McEliece schemes with a quadratic extension. In [COT14], the authors already presented a poly-time attack in this particular case: it applies for the parameters with $m = 2$, but not for the other ones. We want to stress that our attack also works for $m = 2$ and any t ([COT14] does not work in the extreme case $t = 1$). Also, we emphasize that, whilst solving a non-linear system, our attack is actually faster than [COT14] in some cases. For $q = 32$ and $t = 4$, the attack of [COT14] requires 49.5 minutes (using a non-optimized Magma implementation according to the authors). We can mount our attack in several seconds with our techniques.

In practice, we could not solve (in less than two days) the algebraic systems involved when the number of free variables $\#\mathbf{Z}$ exceeds 65. We recall the relation $\#\mathbf{Z} = ((p-1)ms - \#\mathcal{L}_a)\,t$ (for $q = p^s$ and $s > 1$), which should help the designers to scale their parameters. An important remaining open question is to give a precise complexity estimates for the polynomial system solving phase in those cases.

1. available at `http://christianepeters.wordpress.com/publications/tools/`

q	m	t	n	k	$\deg(f)$	Key (kB)	**ISD**	**SSA**	Solving $\mathcal{W}_{q,a}(\mathbf{Z})$, **optimal** a
$32^\star$	2	2	678	554	0	43	$\mathbf{2^{93}}$	$2^{939} \cdot 2^{19}$	8× 0.08 s (#$\mathbf{Z} = 9$)
32	2	4	841	601	0	92	$\mathbf{2^{128}}$	$2^{688} \cdot 2^{38}$	16× 10 s (#$\mathbf{Z} = 36$)
32	2	5	800	505	0	93	$\mathbf{2^{136}}$	$2^{771} \cdot 2^{48}$	20× (2 min 45s) (#$\mathbf{Z} = 40$)
27	3	3	1312	1078	0	45	$\mathbf{2^{113}}$	$2^{6947} \cdot 2^{41}$	15× 3.0 s (#$\mathbf{Z} = 39$)
27	3	4	1407	1095	0	203	$\mathbf{2^{128}}$	$2^{7304} \cdot 2^{55}$	20× (6 min 34 s) (#$\mathbf{Z} = 52$)
$27^\star$	3	4	1312	1000	0	184	$\mathbf{2^{128}}$	$2^{6947} \cdot 2^{55}$	20× (10 min) (#$\mathbf{Z} = 52$)
27	3	5	1700	1310	0	304	$\mathbf{2^{158}}$	$2^{8343} \cdot 2^{69}$	25× (1h 59 min) (#$\mathbf{Z} = 65$)
27	3	5	1800	1410	0	327	$\mathbf{2^{160}}$	$2^{8679} \cdot 2^{69}$	25× (1h 37 min) (#$\mathbf{Z} = 65$)
$16^\star$	3	4	883	703	0	79	$\mathbf{2^{86}}$	$2^{3074} \cdot 2^{46}$	12× (6 min 40s) (#$\mathbf{Z} = 36$)
$16^\star$	3	5	1011	786	0	88	$\mathbf{2^{100}}$	$2^{3296} \cdot 2^{58}$	15× (2h 15 min) (#$\mathbf{Z} = 45$)
16	3	6	1316	1046	0	141	$\mathbf{2^{129}}$	$2^{3703} \cdot 2^{69}$	18× (36h 26 min) (#$\mathbf{Z} = 54$)
$9^\star$	3	6	714	570	0	325	$\mathbf{2^{75}}$	$2^{102} \cdot 2^{54}$	18× (24h 52 min) (#$\mathbf{Z} = 54$)
q	m	t	n	k	$\deg(f)$	Key (kB)	**ISD**	**SSA**	Solving $\mathcal{W}_{q,a}(\mathbf{Z})$, **optimal** a
$32^\star$	2	1	532	406	32	32	$\mathbf{2^{82}}$	$2^{1017} \cdot 2^{335}$	4× 0.02 s (#$\mathbf{Z} = 6$)
$32^\star$	2	2	864	668	36	82	$\mathbf{2^{124}}$	$2^{635} \cdot 2^{384}$	8× 0.09 s (#$\mathbf{Z} = 9$)
32	2	3	852	621	24	90	$\mathbf{2^{130}}$	$2^{663} \cdot 2^{273}$	12× 0.6 s (#$\mathbf{Z} = 18$)
$27^\star$	3	2	1504	1240	36	195	$\mathbf{2^{127}}$	$2^{6121} \cdot 2^{393}$	10× 0.8 s (#$\mathbf{Z} = 26$)
27	3	2	1500	1218	42	204	$\mathbf{2^{128}}$	$2^{5253} \cdot 2^{225}$	10× 0.9 s (#$\mathbf{Z} = 26$)
$25^\star$	3	3	974	719	13	106	$\mathbf{2^{99}}$	$2^{7658} \cdot 2^{546}$	15× (1h 25 min) (#$\mathbf{Z} = 57$)
25	3	3	1206	915	25	155	$\mathbf{2^{117}}$	$2^{7643} \cdot 2^{632}$	15× (1h 2 min) (#$\mathbf{Z} = 57$)
$16^\star$	3	4	908	680	16	78	$\mathbf{2^{93}}$	$2^{3120} \cdot 2^{242}$	12× (3 min 20s) (#$\mathbf{Z} = 36$)
$16^\star$	3	5	1120	853	14	114	$\mathbf{2^{125}}$	$2^{3460} \cdot 2^{230}$	15× (2h 17 min) (#$\mathbf{Z} = 45$)
16	3	6	1328	1010	16	160	$\mathbf{2^{125}}$	$2^{3716} \cdot 2^{265}$	18× (36h 35 min) (#$\mathbf{Z} = 54$)
9	3	6	728	542	14	40	$\mathbf{2^{81}}$	$2^{2759} \cdot 2^{191}$	18× (25h 13 min) (#$\mathbf{Z} = 54$)

Table 7.2 – Practical experiments with Wild McEliece & Incognito parameters. ISD complexities are obtained thanks to Peters' software[1]. SSA attack complexity is given under the form (support enumeration)·(Goppa polynomial enumeration). The $\star$ indicates parameters extracted from the website.

7.4 Conclusion

We proposed an algebraic modelling which allows to exploit the additive structure introduced into the Wild Goppa codes compared to usual Goppa codes. We emphasize that the well-known property $\mathscr{G}(\mathbf{x}, g) = \mathscr{G}(\mathbf{x}, g^2)$ when $g \in \mathbb{F}_{2^m}[z]$ has no square factors shows that binary Goppa codes are almost always wild Goppa codes. Unfortunately, in this special case, the quadratic equations are trivial, the systems $\mathcal{W}_{q,s}(\mathbf{Z})$ contain only linear equations and are strongly underdefined. So, this modelling does not impact at all the security of binary Goppa codes with irreducible Goppa polynomial.

This work focused on Goppa polynomials with at least one factor with multiplicity equal to $q-1$, but we can deduce a warning for McEliece schemes with any multiplicity. Pick $\mathscr{G}(\mathbf{x}, \Gamma)$ a Goppa code with $\Gamma(z) \in \mathbb{F}_{q^m}[z]$. Write Γ as a product of irreducible polynomials $f_i \in \mathbb{F}_{q^m}[z]$: $f = f_1^{m_1} \dots f_s^{m_r}$, where we suppose that the multiplicities have been sorted in decreasing order: $m_1 > \cdots > m_r$. We gather together the factors with same multiplicities, that is we write

$$\Gamma = g_1^{m_1} \dots g_r^{m_r}, \text{ with } g_i = \prod_{\mathrm{mult}(f_j)=m_i} f_j.$$

Let $t = \deg(g_1)$ the support can be recovered by applying our adapted Sidelnikov-Shestakov on the solutions of the system

$$\bigcup_{1 \leqslant u \leqslant m_1} \Big\{ g_{i,0} Z_0^u + \cdots + g_{i,n-1} Z_{n-1}^u = 0 \ \mid \ i \in \{0, \dots, k-1\} \Big\}.$$

Those solutions form a union of vector spaces of dimension $\deg(f)$, so that the systems to solve contain $n - k - \deg(g_1) = m\left(\sum_{1 \leqslant i \leqslant r} m_i \deg(g_i)\right) - \deg(g_1)$ variables. The number of equations is given by $m_1 k = m_1 \left(n - m\left(\sum_{1 \leqslant i \leqslant r} m_i \deg(g_i)\right)\right)$. A good choice of parameters should, in any case, give an underdefined system.

Chapter 8

Toward a Secure Implementation of McEliece Decryption

This chapter presents results from an article written with Mariya Georgieva: *Toward a Secure Implementation of McEliece Decryption*, accepted for publication at *COSADE 2015* ([GdP15]).

The purpose of this chapter is to analyze the security regarding timing attacks of the decryption in McEliece scheme (Algorithm 4) instantiated with a binary Goppa code.

This decryption mainly consists in a decoding, that is, for $\mathbf{m} \in \mathbb{F}_q^k, \mathbf{G} \in \mathbb{F}_q^{k\times n}$ and $\mathbf{e} \in \mathbb{F}_q^n$, to recover $\mathbf{m}$ and $\mathbf{e}$ from $\mathbf{c} = \mathbf{mG} + \mathbf{e}$. We explained in Section 1.3 of Chapter 1 how to perform such decoding when $\mathbf{G}$ is the generator matrix of a binary Goppa code and $\mathbf{e}$ has weight less than or equal to the degree of the Goppa polynomial (Algorithms 2 and 3). When implementing Algorithms 2 and 3, a problem arises in most of the past proposals (*e.g.* in [Bis10, Hey11, Str10a, SWM$^+$10]) because the operation flow of the decryption is strongly influenced by the error vector $\mathbf{e}$, but no information is known about the error vector when starting decryption. From an attacker's point of view, this is a favorable situation. It means that the observed or manipulated device may leak information before any detection of the attack. These security aspects were addressed by various authors, who explained that a device implementing an unprotected decryption is prone to attacks on the messages [SSMS09, AHPT11] and on the key [Str10b, Str13]. Although counter-measures were proposed against some of the leakages, the situation is still unsatisfactory, as it is noticed in the conclusion of [Str13]. In particular, to the best of our knowledge, no decryption algorithm requiring a number of steps independent of the error weight was described.

The work that we present here gathers the different weaknesses revealed in [SSMS09, AHPT11, Str10b, Str13]. We detail in Section 8.1 the existing message and key-recovery attacks exploiting timing leakages. In particular, we explain the attacks of Strenzke and show that they can be extended to bypass the counter-measure of [Str10b]. All these attacks targeted only one of the two known methods for decoding a binary Goppa code (namely Patterson algorithm, Algorithm 3). Along with our presentation, we evaluate how/if those threats transpose to the other decoding method (*i.e.* the alternant decoder, Algorithm 2). Then, we describe in Section 8.2 an Extended Euclidean Algorithm (EEA) tailored for the alternant decoder which has a flow of operations independent of the error vectors (Algorithm 16). The new algorithm was inspired by a work of Berlekamp [BSP94]. We explain step-by-step the construction of the algorithm, and provide completeness proofs in Section 8.3 (which we could not find in the literature).

To ease the reading of this chapter, we recall here the Algorithms allowing to perform the decoding on binary Goppa codes. $\mathbf{G}$ generates a binary Goppa code $\mathscr{G}(\mathbf{x}, g)$ (Defintion 1.22) with $\deg(g) = t$ and $w \leqslant t$ with $\mathbf{x} \in \mathbb{F}_{q^m}^n$ and $g(z) \in \mathbb{F}_{q^m}[z]$ of degree t.

Input: $\mathbf{c} = \mathbf{mG} + \mathbf{e}, \mathbf{x}$ and $g(z)$.
Output: $\mathbf{e} = (0, \dots, e_{i_1}, \dots, 0, e_{i_w}, 0, \dots)$.

Polynomial syndrome:

$$S_{Alt,\mathbf{e}}(z) = \sum_{\ell=0}^{2t-1} \left(\sum_{i=0}^{n-1} c_i g(x_i)^{-2} x_i^{\ell} \right) z^{\ell}.$$

Polynomials to be recovered:

$$\sigma_{inv,\mathbf{e}}(z) = \prod_{j=1}^{w} (1 - z x_{i_j}),$$

$$\omega_{inv,\mathbf{e}}(z) = \sum_{j=1}^{w} e_{i_j} g(x_{i_j})^{-1} \prod_{\substack{s=1 \\ s \neq j}}^{w} (1 - z x_{i_s}).$$

Key equation:
$(\sigma_{inv,\mathbf{e}}, \omega_{inv,\mathbf{e}})$ unique solution of

$$\begin{cases} \omega_{inv,\mathbf{e}}(z) = \sigma_{inv}(z) S_{Alt,\mathbf{e}}(z) \mod z^{2t}, \\ \deg(\sigma_{inv}) \leqslant \lfloor t/2 \rfloor, \deg(\omega_{inv}) \leqslant \lfloor t/2 \rfloor - 1. \end{cases}$$

Resolution
$\mathrm{EEA}(z^{2t}, S_{Alt,\mathbf{e}}, t) = (\mu \sigma_{inv}, (-1)^N \mu \omega_{inv})$, for some $\mu \in \mathbb{F}_{q^m}^*, N \geqslant 0$.

Error recovery
$\sigma_{\mathbf{e}}(z) = z^w \sigma_{inv}(1/z)$.
Find the roots of $\sigma_{\mathbf{e}}$.

Figure 8.1 – Alternant Decoder

Input: $\mathbf{c} = \mathbf{mG} + \mathbf{e}, \mathbf{x}$ and $g(z)$.
Output: $\mathbf{e} = (0, \dots, e_{i_1}, \dots, 0, e_{i_w}, 0, \dots)$.

Polynomial syndrome:

$$S_{Gop,\mathbf{e}}(z) = \sum_{i=0}^{n-1} \frac{c_i}{z - x_i} \mod g(z).$$

Polynomials to be recovered:

$$\sigma_{\mathbf{e}}(z) = \prod_{j=1}^{w} (z - x_{i_j}),$$

$$\omega_{\mathbf{e}}(z) = \sum_{j=1}^{w} \prod_{\substack{s=1 \\ s \neq j}}^{w} (z - x_{i_s}).$$

Key equation:
(σ_1, σ_2) unique solution of

$$\begin{cases} \tau(z)\sigma_2(z) = \sigma_1(z) \mod g(z), \\ \deg(\sigma_1) \leqslant \lfloor t/2 \rfloor, \deg(\sigma_2) \leqslant \lfloor t/2 \rfloor - 1, \end{cases}$$

with $\tau(z) = \sqrt{S_{Gop,\mathbf{e}}(z)^{-1} + z} \mod g(z)$.

Resolution
1.$\mathrm{EEA}(g(z), S_{Gop,\mathbf{e}}(z), 0) = (., S_{Gop,\mathbf{e}}(z)^{-1})$,
2.$\mathrm{EEA}(g(z), \tau, \lfloor t/2 \rfloor) = (\sigma_1, \sigma_2)$,

Error recovery
$\sigma_{\mathbf{e}}(z) = \sigma_1(z)^2 + z\sigma_2(z)^2$,$\omega_{\mathbf{e}} = \sigma_{\mathbf{e}} S_{\mathbf{e}} \mod g$.
Find the roots of $\sigma_{\mathbf{e}}$.

Figure 8.2 – Patterson Algorithm

8.1 Timing Attacks against McEliece Decryption

The attacks that we review aim at recovering information on an encrypted message or on the secret key by observing the influence of the error weight on the behaviour of the decryption algorithm as implemented in Patterson decoding (Algorithm 8.2). Of particular interest, it contains an extended Euclidean algorithm (EEA) which performs successive Euclidean divisions as in Algorithm 1. As the Alternant decoder also contains an EEA, we apply the different attacks proposed to both decoding algorithms in order to determine which one is the most resistant.

8.1.1 Plaintext-Recovery Attacks

The authors of [SSMS09, AHPT11, Str10b] described attacks using a common framework. They exploit a decryption oracle to recover the plaintext from an encrypted message $\mathbf{c}$: the attacker can request decryption of any message $\mathbf{c}' \neq \mathbf{c}$ and observe the execution of the decryption.

Generic Attack Scenario

The common framework is summed up in Algorithm 9.

Algorithm 9 Framework for message-recovery attacks on a decryption device.

INPUT: A valid ciphertext $\mathbf{c} = \mathbf{m}\mathbf{G}_{pub} + \mathbf{e}$, a decryption device $\mathcal{D}$.
OUTPUT: The error vector $\mathbf{e}$ and plaintext $\mathbf{m}$.

1: **for** $i = 0, \dots, n-1$ **do**
2: Modify $\mathbf{c}$ into $\mathbf{c}^{\star i} = \mathbf{c} + (0, \dots, 0, \underbrace{1}_{i-\text{th bit}}, 0, \dots)$.
3: Request decryption $\mathcal{D}(\mathbf{c}^{\star i})$.
4: Deduce by timing analysis or power consumption of $\mathcal{D}$ whether $e_i = 0$ or $e_i = 1$.
5: **end for**
6: Solve the linear system with unknown $\mathbf{m}$: $\mathbf{m}\mathbf{G}_{pub} = \mathbf{c} + \mathbf{e}$.
7: **return** Plaintext $\mathbf{m}$.

We describe in Algorithm 9 an attack against a McEliece encryption scheme. The same framework is applicable against Niederreiter encryption when a public encryption key $\mathbf{H}_{pub}$ is known. It suffices to replace Step 2 by

2: Modify $\mathbf{c}$ into $\mathbf{c}^{\star i} = \mathbf{c} + \mathbf{H}_{pub}^{(i)}$, where $\mathbf{H}_{pub}^{(i)}$ denotes the i-th row of $\mathbf{H}_{pub}$.

Remark 8.1. *In some variants of McEliece encryption, called CCA2 conversions [KI01, Per12b, DDMQN12], a link is imposed between the message and the error. Ciphertext manipulation is detected thanks to a test on the validity of the error and the algorithm outputs a failure: the attacker only checking the output does not recover information except that the ciphertext was not valid. However, an attacker observing the behaviour of the computation can deduce the error bits anyway. As the test on the error is performed* after *the decoding, it is mandatory to eliminate the leakages in the decoding even for CCA2 conversions.*

The first method to detect variations of the error weight was explained in [STM+08] and refined in [AHPT11]. It dates back to 2008 and focuses on the determination of the roots of the error locator polynomial $\sigma_{\mathbf{e}}(z)$ (Figure 8.2). The authors of [AHPT11] and [STM+08]

propose convincing counter-measures against this weakness (see [AHPT11][Algorithm 4]). Indeed, when starting the root-finding step, errors of weight lower than t can be detected thanks to the low degree of the error locator polynomial. This makes this step rather simple to protect. Unfortunately, this is not sufficient to discard attacks following the framework of Algorithm 9. The reason is that, as first noticed in [SSMS09], the EEA determining $\sigma_{\mathbf{e}}$ also has an execution time depending on the error weight. Now we focus on the EEA step which is still problematic.

8.1.1.1 Timing attacks on the root-finding step.

The method used in [STM+08, AHPT11] dates back to 2008 and focuses on the determination of the roots of the error locator polynomial $\sigma_{\mathbf{e}}(z)$ (Algorithm 8.2). The idea is the following. Let's apply the framework of Algorithm 9 and look at the i-th iteration of the **for** loop. We consider binary codes, so that the secret bit e_i is equal to 0 or 1.

1. **If** $e_i = 1$**.** The twisted ciphertext $\mathbf{c}^{\star i}$ satisfies $\mathbf{c}^{\star i} = \mathbf{mG}_{pub} + \mathbf{e}^{\star i}$ with $w_H(\mathbf{e}^{\star i}) = w_H(\mathbf{e}) - 1$. After solving the key equation, the error-locator polynomial obtained has degree $w_H(\mathbf{e}) - 1$. Determining its roots may be done either by evaluating it at all the support points (often $\mathbb{F}_{2^m}$) or computing its greatest common divisor with the field equation $z^{2^m} - z$. These operations take a time which depends on the degree of the error locator polynomial. On an unprotected implementation, a shorter root-finding step implies a lower degree error locator polynomial, so it implies that $e_i = 1$.
2. **If** $e_i = 0$**.** The twisted ciphertext $\mathbf{c}^{\star i}$ satisfies $\mathbf{c}^{\star i} = \mathbf{mG}_{pub} + \mathbf{e}^{\star i}$ with $w_H(\mathbf{e}^{\star i}) = w_H(\mathbf{e}) + 1$. For a correct ciphertext, the number of errors is set as high as possible, so $\mathbf{c}^{\star i}$ is not a decodable message anymore. Decoding processes and EEA computes a polynomial $\sigma(z)$ which is still of degree t but $\sigma(z)$ does not split over $\mathbb{F}_{2^m}$ any more. Root-finding by polynomial evaluation or gcd computation both allow to detect this case in the same time as with a correct error locator polynomial.

These two different behaviors can be distinguished by measuring time execution, so that an attacker can know whether $e_i = 1$ or $e_i = 0$ and recover $\mathbf{e}$.

Counter-Measure for the root-finding step.
The authors of [AHPT11] and [STM+08] propose a counter-measure against this vulnerability. The idea in both case is to make the root-finding step's execution time independant of the weight of the error vector. To do so, they propose an extra-step to transform the output of the EEA algorithm into a degree t polynomial. It is crucial that this step's execution time be independant of the weight of the error vector. A convincing proposition is made in [AHPT11][Algorithm 4].

We note that, when starting the root-finding step, the leaking case ($e_i = 1$) has already been detected thanks to the low degree of the error locator polynomial. This makes this step rather simple to protect. Unfortunately, this is not sufficient to discard attacks following the framework of Algorithm 9. The reason is that, as first noticed in [SSMS09], theEEA determining the error locator polynomial also has an execution time depending on the error weight. This is the object of the next paragraph.

8.1.1.2 Exploitation of EEA leakages.

The authors of [SSMS09] focused on the second EEA of Patterson algorithm (Fig. 8.2 and Paragraph 1.3.2 of Chapter 1). Implemented as in Algorithm 1, the number of iterations

in the **while** loop depends on the weight of the error. To do so, they show that the output $\sigma_1(z), \sigma_2(z)$ satisfy the relations:

$$\deg(\sigma_2(z)) = \sum_{i=1}^{N} \deg(q_i) \tag{8.1}$$

$$\deg(\sigma_1(z)^2 + z\sigma_2(z)^2) = w_H(\mathbf{e}) \tag{8.2}$$

Thanks to relations (8.1) and (8.2), they deduce that, assuming that all the Euclidean divisions in the second EEA have a quotient of degree one (which happens with probability $(1-2^{-m})^N$), the number N of iterations in the **while** loop varies as in Table 8.1. These variations of N can be detected either by measuring time execution or counting the number of patterns on a power consumption trace. Thus, they provide a successful tool to perform Algorithm 9 and recover a plaintext.

Counter-Measure for the Second EEA leakage.

To protect against the previous attack, the authors of [SSMS09] propose to check the degrees of the polynomials processed by the EEA during its execution instead of checking only the output. We recall that, in Patterson's decoding, the error locator polynomial is split into even and odd part: $\sigma_{\mathbf{e}}(z) = \sigma_1(z)^2 + z\sigma_2(z)^2$. [SSMS09] gives a precise description of the degrees to be expected for $\sigma_1(z)$ and $\sigma_2(z)$ according to the weight of the error, recalled in Table 8.1.

	$t = 2t'$			$t = 2t' + 1$		
	N	$\deg(\sigma_1)$	$\deg(\sigma_2)$	N	$\deg(\sigma_1)$	$\deg(\sigma_2)$
$w_H(\mathbf{e}) = t+1$	$\leqslant t'-1$	$= t'$	$\leqslant t'-1$	$\leqslant t'$	$\leqslant t'$	$= t'$
$w_H(\mathbf{e}) = t$	$\leqslant t'-1$	$= t'$	$\leqslant t'-1$	$= t'$	$\leqslant t'-1$	$= t'$
$w_H(\mathbf{e}) = t-1$	$= t'-1$	$\leqslant t'-1$	$= t'-1$	$\leqslant t'-1$	$= t'-1$	$\leqslant t'-1$
$w_H(\mathbf{e}) = t-2$	$\leqslant t'-2$	$= t'-1$	$\leqslant t'-2$	$= t'-1$	$\leqslant t'-2$	$\leqslant t'-1$

Table 8.1 – Degrees of the output polynomials of $\mathrm{EEA}(g(z), \tau(z), \lfloor t/2 \rfloor)$.

They deduce Algorithm 10, a modified EEA which detects the ciphertext manipulation and completes the computation to take same execution time both on the ciphertext $\mathbf{c}$ and on the twisted $\mathbf{c}^{\star i}$. With this version, the authors claim to obtain a decryption time taking a constant number of clock cycles. This would discard a timing attack. However, no power consumption trace is analyzed. It remains unclear whether the extra manipulation in Step 9 of Algorithm 10, whose execution implies that $e_i = 1$, is undetectable. In particular, each **while** iteration will not have the same execution time. Fakely executing those steps at each iteration would have a serious impact on the performances.

EEA leakages in the Alternant Decoder.

We adapt the framework of Algorithm 9 to the alternant decoder. The alternant decoder(Algorithm 8.1 and Paragraph 1.3.1 of Chapter 1), as Patterson one, resorts to an EEA

Algorithm 10 Protected EEA for Patterson decoding

Beginning as is Algorithm 1.

7: $u_i \leftarrow u_{i-2}(z) - q_i(z)u_{i-1}(z)$

CASE $t = 2t'$.

8: **if** $\deg(r_i) < t'$ **then**
9: Manipulate r_i so that $\deg(r_i) = \deg(r_{i-1}) - 1$.
10: **end if**

CASE $t = 2t' + 1$.

8: **if** $\deg(r_i) \leqslant t'$ and $\deg(v_i) < t'$ **then**
9: Manipulate r_i so that $\deg(r_i) = \deg(r_{i-1}) - 1$.
10: **end if**

End as is Algorithm 1.

prone to leak information when the error weight varies. The equivalent of Relations (8.1) and (8.2) in this context is

$$\deg(\sigma_{inv}(z)) = \sum_{i=1}^{N} \deg(q_i) = \begin{cases} w_H(\mathbf{e}) & \text{if } 0 \notin \mathbf{x} \\ w_H(\mathbf{e}) - 1 & \text{if } 0 \in \mathbf{x}. \end{cases} \tag{8.3}$$

We sum in Table 8.2 the link of the degree of the output polynomial of the EEA in Algorithm 8.1 with the weight of the error vector (α denotes the position of the support such that $x_\alpha = 0$).

	$\deg(\sigma_{inv})$ if $e_\alpha = 0$	$\deg(\sigma_{inv})$ if $e_\alpha = 1$
$w_H(\mathbf{e}) = t$	t	$t-1$
$w_H(\mathbf{e}) = t + 1 (e_i = 0)$	t	$t-1$
$w_H(\mathbf{e}) = t - 1 (e_i = 1)$	$t-1$	$t-2$

Table 8.2 – Degrees of the output polynomials of $\mathrm{EEA}(z^t, S_{\mathbf{e}}(z), \lfloor t/2 \rfloor)$.

We recall that the error locator polynomial is deduced from $\sigma_{inv}(z)$ when the weight of the error vector is known (see Remark 1.29 of Chapter 1). Therefore, in this case, looking at the degree of σ_{inv} does not distinguish manipulated ciphertexts from correct ones, and the EEA cannot be correctly protected by this method.

First Counter-Measure for the EEA in the Alternant Decoder

Building up on the counter-measure for Patterson decoding described in [SSMS09], we propose the following adaptation (Algorithm 11) to the alternant decoder. It always detects ciphertext manipulation provided that 0 is not an element of the support, and somehow restores a usual behavior of the EEA (that is, that of a valid ciphertext). The final output will not be the correct plaintext, but this is not a problem as long as the attacker cannot extract information from this result. However, we note that this protection has the same drawbacks as its Patterson equivalent: each **while** execution does not have same execution time.

Algorithm 11 Protected EEA for Alternant decoder

Beginning as is Algorithm 1.

7: $v_i \leftarrow v_{i-2}(z) - q_i(z)v_{i-1}(z)$
8: **if** $\deg(r_i) < t$ **then**
9: Manipulate r_i so that $\deg(r_i) = \deg(r_{i-1}) - 1$
10: **end if**

End as is Algorithm 1.

8.1.2 Secret decryption key recovery attacks

We address a different kind of physical attack initiated by Strenzke in [Str10b, Str13] against McEliece encryption using Patterson decoding. It aims at recovering the secret key.

Generic attack scenario.

The attack scenario is the following. The attacker has access to a decryption device $\mathcal{D}$ on which he can perform physical measurements. He also knows a public encryption key, so that he can generate codewords with errors of his choice. By observing the decryption phase (more precisely, the EEA execution), Strenzke shows that one can deduce information on the support elements corresponding to the error positions. Roughly, the reason is that when a polynomial condition on those elements is satisfied, the number of iterations of the **while** loop in Algorithm 1 is reduced compared to the average number of iterations necessary to perform the EEA for error vectors of same weight. The attack consists in scanning a lot of error positions and collect sufficiently many polynomial relations so that the algebraic system obtained can be solved.

We sum up in Algorithm 12 (on page 143) the global attack framework arising from [Str13].

Algorithm 12 Framework for key-recovery attacks on a decryption device.

INPUT: A decryption device $\mathcal{D}$, public encryption key $\mathbf{G}_{pub}$.
OUTPUT: The secret support $\mathbf{x}$.

1: **for** w well-chosen error weights **do**
2: **for** $(i_1, \ldots, i_w)$ subset of $\{0, \ldots, n-1\}$ **do**
3: Pick a low-weight error vector $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_w}, \ldots, 0)$ with $w_H(\mathbf{e}) = w$.
4: Request decryption $\mathcal{D}(\mathbf{e})$.
5: Perform timing or power consumption analysis of $\mathcal{D}(\mathbf{e})$.
6: **if** EEA execution is faster than average (more precise conditions in the rest of this Section) **then**
7: Deduce a polynomial condition on $x_{i_1}, \ldots, x_{i_w}$ (P_w is a polynomial depending only on w):

$$P_w(x_{i_1}, \ldots, x_{i_w}) = 0 \tag{8.4}$$

8: **end if**
9: **end for**
10: **end for**
11: Solve the non-linear system of all the collected equations (8.4).
12: **return** Secret support $\mathbf{x} = (x_0, \ldots, x_{n-1})$.

In practice, the polynomials P_w will be elementary symmetric polynomials of the form,

for an error $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_w}, \ldots, 0)$ with $w_H(\mathbf{e}) = w$ and $j \geqslant 0$:

$$\omega_j(\mathbf{e}) = \sum_{1 \leqslant \ell_1 < \cdots < \ell_j \leqslant w} x_{i_{\ell_1}} \ldots x_{i_{\ell_j}}.$$

That is, $\omega_j(\mathbf{e})$ is the evaluation of the jth elementary symmetric polynomial in w variables in $(x_{i_1}, \ldots, x_{i_w})$.

State-of-the-art

More precisely, Strenzke uses errors of weights $w = 1, w = 4$ and $w = 6$. For $w = 6$, errors such that Equation (8.4) is satisfied are harder to find than for $w = 4$. For this reason, his strategy consists in collecting as many Equations (8.4) with $w = 1$ and $w = 4$ as possible. He obtains a linear system of rank $n - m$ (in some cases $n - m - 1$) in the n elements of the support. Then, he selects subsets of errors of weight $w = 6$ to look for Equations (8.4). These subsets are chosen so as help the polynomial system solving. According to Strenzke, for an encryption scheme with parameters $m = 10, n = 2^m, t = 40$, it takes about 15,000,000 decryption queries to collect enough equations and 28 hours to solve the algebraic system. Eventually, the full secret support $\mathbf{x}$ is recovered by the attacker, and then the secret $\mathbf{y}$ thanks to Lemma 4.1 and the Goppa polynomial by interpolation.

8.1.2.1 First example of leakage exploitable by Framework 12.

The first attack resorting to the method of Algorithm 12 was proposed by Strenzke in [Str10b]. It focuses on the second EEA of Patterson decoder with errors of weight $w = 4$. In this case, $S_{\mathbf{e}}(z) = \sum_{j=1}^{4} \frac{1}{z - x_{i_j}} = \frac{\omega_{\mathbf{e}}(z)}{\sigma_{\mathbf{e}}(z)}$, and

$$\omega_{\mathbf{e}}(z) = \underbrace{(x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4})}_{\omega_1(\mathbf{e})} z^2 + \underbrace{x_{i_1}x_{i_2}x_{i_3} + x_{i_1}x_{i_2}x_{i_4} + x_{i_1}x_{i_3}x_{i_4} + x_{i_2}x_{i_3}x_{i_4}}_{\omega_3(\mathbf{e})}.$$

If $\omega_1(\mathbf{e}) = 0$, then $S_{\mathbf{e}}(z) = \frac{\omega_3(\mathbf{e})}{\sigma_{\mathbf{e}}(z)}$, and $S_{\mathbf{e}}^{-1} \mod g = \omega_3(\mathbf{e})^{-1}\sigma_{\mathbf{e}}(z)$ therefore $\tau(z) = \sqrt{S_{\mathbf{e}}^{-1}(z) + z} \mod g(z) = \sqrt{\omega_3(\mathbf{e})^{-1}\sigma_{\mathbf{e}}(z) + z}$ and $\tau(z)$ has degree lower than $\lfloor t/2 \rfloor$ (for $w = 4$ we have $\deg(\tau(z)) = 2$). As a consequence, the **while** test in EEA$(g(z), \tau(z), \lfloor t/2 \rfloor)$ is never fulfilled and the number of iterations N is equal to 0. When $\omega_1(\mathbf{e}) \neq 0$, $\deg(\tau(z)) > \lfloor t/2 \rfloor$ with overwhelming probability ($\tau(z)$ is a reduction modulo a polynomial of degree t), so that $N > 0$. This allows to collect many equations of the form

$$x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4} = 0.$$

As Strenzke explains, the final system's rank never exceeds $n - m$. So it is not sufficient in practice to recover the private key and one has to assume that some other parts of private elements are required. Still, he proposes a counter-measure to avoid this information leakage.

Counter-Measure to protect Second EEA by Strenzke.

Strenzke proposes to detect the polynomials $\tau(z)$ leading to this leakage by checking if $\deg(\tau(z)) < \lfloor t/2 \rfloor$. This can be done just after the determination of $\tau(z)$. If so, manipulate $\tau(z)$ so that is has degree $t - 1$.

This counter-measure avoids leaking information only in the second EEA, only when decoding errors of weight 4. Exploitable leakages remain, as shown in the next paragraph.

Algorithm 13 Patterson decoding with Strenzke's counter-measure against weight 4 errors leakages

INPUT: $n-k$-bit syndrome $\mathbf{s} = \mathbf{H}\mathbf{e}^T$ with $w_H(\mathbf{e}) \leqslant \lfloor t \rfloor$, private key $(\mathbf{x}, g(z))$
OUTPUT: The error vector $\mathbf{e}$.

1: POLYNOMIAL SYNDROME DETERMINATION: Idem Fig. 8.2.
COMPUTATION OF $\tau(z)$:
2: Find $f(z)$ such that $f(z)S_{\mathbf{e}}(z) = 1 \mod g(z)$ by $\cdot$, $f(z) = \mathrm{EEA}(g(z), S_{\mathbf{e}}(z), 0)$
3: Set $\tau(z) = \sqrt{f(z) + z}$.
4: **if** $\deg(\tau(z)) < \lfloor t/2 \rfloor$ **then**
5: Manipulate $\tau(z)$ so that $\deg(\tau(z)) = t - 1$.
6: **end if**
7: ERROR LOCATOR POLYNOMIAL DETERMINATION & ERROR VECTOR DETERMINATION Idem Fig. 8.2.

8.1.2.2 Leakage in the first EEA of Patterson Decoding.

In order to complete the attack initiated in [Str10b], Strenzke proposed in [Str13] to apply Algorithm 12 by focusing on time leakages in both EEA's of Patterson decoding. In [Str13][Corollary 1], he gives the number of iterations of the **while** loop in the first EEA. We recall it here, and complete it with the analogous result for the second EEA (which we could not find in [Str13]).

Lemma 8.2. *Let $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$ be a binary Goppa code and $S_{\mathbf{e}}(z)$ the polynomial syndrome associated to an error $\mathbf{e}$ with $w_H(\mathbf{e}) \leqslant \deg(g)/2 - 1$. Write $S_{\mathbf{e}}(z) = \frac{\omega_{\mathbf{e}}(z)}{\sigma_{\mathbf{e}}(z)} \mod g(z)$. Let N_I and N_K be the number of iterations of the* **while** *loop respectively in* $\mathrm{EEA}(g(z), S_{\mathbf{e}}(z), 0)$ *and* $\mathrm{EEA}(g(z), \tau(z), \lfloor t/2 \rfloor)$. *Then*

$$N_I \leqslant \deg(\omega_{\mathbf{e}}(z)) + \deg(\sigma_{\mathbf{e}}(z)) \text{ and } N_K \leqslant \deg(\omega_{\mathbf{e}}(z))/2. \tag{8.5}$$

Proof. The result on N_I is proved in [Str13][Corollary 1]. Regarding N_K, observe that v_0 has degree 0 and $v_{N_K} = \sigma_2(z)$ has degree $\deg(\omega_{\mathbf{e}})/2$ (since by derivating the relation $\sigma = \sigma_1^2 + z\sigma_2^2$ we obtain $\omega_{\mathbf{e}} = \sigma_2^2$). As the degrees are raised at least by one at each iteration, we obtain $N_K \leqslant \deg(\omega_{\mathbf{e}})/2$. □

Let's apply this small weight error vectors. Note that, for any error $\mathbf{e}$, $\deg(\sigma_{\mathbf{e}}) = w_H(\mathbf{e})$.

Exploiting Leakages of errors with weight $w = 4$.
Pick $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_4}, \ldots, 0)$. We know that $\omega_{\mathbf{e}}(z) = \omega_1(\mathbf{e})z^2 + \omega_3(\mathbf{e})$. Lemma 8.2 gives the number N_I of iterations in $\mathrm{EEA}(g(z), S_{\mathbf{e}}(z), 0)$ satisfies

$$\begin{aligned} x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4} \neq 0 &\implies N_I = 6, \\ x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4} = 0 &\implies N_I = 4. \end{aligned}$$

Therefore, even if the second EEA has been protected with Strenzke's counter-measure, errors of weight $w = 4$ leak the same information in the first EEA. This does not give enough equations. To complete the algebraic system, the idea is to use error weights $w = 6$.

Exploiting Leakages of errors with weight $w = 6$.
For $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_6}, \ldots, 0)$, we develop $S_{Gop,\mathbf{e}}(z)$:

$$S_{Gop,\mathbf{e}}(z) = \frac{\omega_1(\mathbf{e})z^4 + \omega_3(\mathbf{e})z^2 + \omega_5(\mathbf{e})}{\sigma_{\mathbf{e}}(z)}.$$

Strenzke's purpose is to detect for which $\mathbf{e}$ is holds that $\omega_3(\mathbf{e}) = \omega_1(\mathbf{e}) = 0$. These cases are exactly those with $S_{\mathbf{e}}(z)^{-1} = \omega_5(\mathbf{e})^{-1}\sigma_{\mathbf{e}}(z)$ and hence $\deg(\tau(z)) < \lfloor t/2 \rfloor$, so that the number of iterations in the second EEA is 0 **provided that Strenzke's counter-measure is not applied**. This is a somehow surprising proposition, since this criterion can be rendered useless by a counter-measure already proposed by the same author.

		EEA$(g, S_{\mathbf{e}}, 0)$	EEA$(g, \tau, \lfloor t/2 \rfloor)$
$w_H(\mathbf{e}) = 4$	$\omega_1(\mathbf{e}) \neq 0$	$N_I \leqslant 6$	$N_K \leqslant 1$
	$\omega_1(\mathbf{e}) = 0$	$N_I \leqslant 4^*$	$N_K = 0^+$ CM $\deg(\tau) < \lfloor t/2 \rfloor^+$
$w_H(\mathbf{e}) = 6$	$\omega_1(\mathbf{e}) \neq 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 10$	$N_K \leqslant 2$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 8$	$N_K \leqslant 1$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) = 0$	$N_I \leqslant 6$	$N_K = 0^*$ CM $\deg(\tau) < \lfloor t/2 \rfloor$
$w_H(\mathbf{e}) = 2w'$	$\omega_1(\mathbf{e}) \neq 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 4w' - 2$	$N_K \leqslant w' - 1$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 4w' - 4$	$N_K \leqslant w' - 2$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) = 0$	$N_I \leqslant 4w' - 6$	$N_K \leqslant w' - 3$

Table 8.3 – Overview of small- error-weight message attacks. Cases marked with a * or a $^+$ are proposed resp. in [Str10b] and [Str13].

8.1.2.3 Combination of first and second EEA.

When using error weights $w \geqslant 6$, the attacker will encounter problems due to the fact that all the values given in 8.3 are only bounds (except in the cases $N \leqslant 0$). Indeed, it may happen that one of the Euclidean divisions entails a degree fall greater than 1 independently of the degree of $\omega_{\mathbf{e}}$.

For example, with $w = 6$, the attacker may observe $N_K = 1$ whereas $\omega_1(\mathbf{e})$ is not zero. This remark leads Strenzke to discard those cases for an attack as long as no way of distinguishing thoses cases is found. We propose such distinguisher, by using N_I to determine if $\omega_1(\mathbf{e})$ is zero, as $\omega_1(\mathbf{e}) = 0$ implies $N_I \leqslant 8$. Indeed, an attacker observing the errors $\mathbf{e}$ with $(N_I, N_K) = (10, 1)$ can conclude that $\omega_1(\mathbf{e}) \neq 0$ (*cf.* Table 8.3). We may have $(N_I, N_K) = (8, 1)$ when $\omega_1(\mathbf{e}) \neq 0$ if three cancellations occur in the 12 intermediate polynomials, which has probability $p_3 = \binom{12}{3} 2^{-3m}(1 - 2^{-m})^9 \approx 2.10^{-7}$ for $m = 10$ (we model the leading coefficients as random elements of $\mathbb{F}_{2^m}$). When sampling x error vectors, we expect

to find p_3x such misleading cases. With the numbers of samples from [Str13][Table 2], the probability to find one is not negligible. If at least one wrong equation is deduced, the system to solve has no solution and the attack fails. We propose to avoid this problem by using errors with $w \geqslant 8$.

Example of leakage exploitation of an error of weight $w = 8$.

We sampled randomly 10,000,000 errors $\mathbf{e}$ of weight 8 and collected the couples (N_I, N_K) in Table 8.4. When $w_H(\mathbf{e}) = 8$, there are more possibilities than with $w = 6$. Samples with $(N_I \leqslant 12, N_K \leqslant 2)$ do not necessarily have $\omega_1(\mathbf{e}) = 0$: this happens with probability $p'_3 = \binom{17}{3} 2^{-3m}(1 - 2^{-m})^{14} \approx 6.10^{-7}$ for $m = 10$ (we found 3). In particular, the case marked with a * in Table 8.4 would make the attacker to think erroneously that the corresponding error vector satisfies $\omega_1(\mathbf{e}) = 0$.

However, the number of parasitic cancellations necessary to provide values (N_I, N_K) compatible with $(\omega_1(\mathbf{e}), \omega_3(\mathbf{e})) = (0, 0)$ is 6, which happens with probability $p'_3 = \binom{17}{6} 2^{-6m}(1 - 2^{-m})^{11} \approx 10^{-14}$ for $m = 10$. If $\omega_1(\mathbf{e}) = 0$ but $\omega_3(\mathbf{e}) \neq 0$, then a couple $(10, 1)$ is found if 3 cancellations occur. This has probability $2^{-m}p'_3 \approx 6.10^{-10}$ (as ω_1 takes all the values of $\mathbb{F}_{2^m}$ with same probability). Therefore, we are able to say without ambiguity when $(\omega_1(\mathbf{e}), \omega_3(\mathbf{e})) = (0, 0)$ on a considerable amount of samples. We deduce from our samples 10 equations $\omega_1(\mathbf{e}) = 0$ which are correct with probability $(1 - 10^{-7})$ and 10 equations $\omega_3(\mathbf{e}) = 0$ correct with probability $(1 - 10^{-3})$.

	No parasitic cancellation	1 parasitic cancellation	2 parasitic cancellations	3 parasitic cancellations
$\omega_1(\mathbf{e}) \neq 0$ $\omega_3(\mathbf{e}) \neq 0$	(14,3): 9855087	(13,3): 115439 (14,2): 18916	(12,3): 614 (13,2): 248 (14,1): 8	(12,2): 1 * (11,3): 2
$\omega_1(\mathbf{e}) = 0$ $\omega_3(\mathbf{e}) \neq 0$	(12,2): 9570	(11,2): 96 (12,1): 8	(10,2): 0 (11,1): 0	
$\omega_1(\mathbf{e}) = 0$ $\omega_3(\mathbf{e}) = 0$	(10,1): 10	(9,1): 0	(8,1): 0	

Table 8.4 – Number of samples for each (N_I, N_K) for 10,000,000 samples error vectors of weight $w = 8$. Code parameters: $m = 10, n = 2^m, t = 40$. See text for explanation on the case with *.

To conclude, although our method requires more samples than the previous one (around 10^9 to collect some thousands of equations with ω_1, and dozens with ω_3), we showed that it is possible to recover information on the support even if the counter-measure $\deg(\tau) < \lfloor t/2 \rfloor$ is implemented.

8.1.2.4 Small weight error messages in Alternant decoder.

We determine if an attacker can retrieve any information by applying Algorithm 12 if the Alternant decoder is implemented. Lemma 8.3, which is analogous to Lemma 8.2, analyses the impact of small weight error messages on the EEA.

Lemma 8.3. *Let* $\mathbf{e}$ *be an error with* $w_H(\mathbf{e}) \leqslant t$. *Then* $S_{Alt,\mathbf{e}}(z) = \frac{\omega_{inv,\mathbf{e}}(z)}{\sigma_{inv,\mathbf{e}}(z)} \mod z^{2t}$ *and the*

number of iterations N of the **while** *loop of the Alternant decoder in the EEA satisfies*

$$N \leqslant N_{max} = \min(\deg(\sigma_{inv,\mathbf{e}}), \deg(S_{Alt,\mathbf{e}}) - \deg(\omega_{inv})) \tag{8.6}$$

Proof. We observe in the execution of $\mathrm{EEA}(z^{2t}, S_{Alt,\mathbf{e}}, t)$. We have $u_0(z) = 0$ and $u_N(z) = \sigma_{inv,\mathbf{e}}(z)$. Since $\deg(u_{i+1}) \geqslant \deg(u_i) + 1$, it holds that $N \leqslant \deg(\sigma_{inv,\mathbf{e}})$. Regarding $(r(z))$, we have $r_0(z) = S_{Alt,\mathbf{e}}(z)$ and $r_N(z) = \omega_{inv}(z)$ so $N \leqslant \deg(S_{Alt,\mathbf{e}}) - \deg(\omega_{inv})$. □

Specific case of weight 1 errors.
If $w = 1$, we always have $\deg(\omega_{inv}) = 0$ and $\deg(\sigma_{inv}) = 1$ *except* if the error is positioned in the zero element of the support. Indeed, in this case, the polynomial syndrome is a constant: $S_{\mathbf{e}}(z) = \frac{1}{g(0)^2}$ and the **while** loop is never executed.

Exploiting error weights $w > 1$.
We suppose that no error occurred in the zero element of the support so that $\deg(\sigma_{inv}) = w_H(\mathbf{e})$ always holds (the coefficient of z^w in σ_{inv} is $x_{i_1} \dots x_{i_w}$). Therefore, faster decryptions indicate the cancellation of a leading coefficient in the intermediate values, but in the alternant decoder we found no way of determining which intermediate value was concerned. If by any chance a power analysis can ensure that it is the first intermediate polynomial (that is, the syndrome polynomial $S_{Alt,\mathbf{e}}(z)$) that has a degree smaller than expected, then the information recovered would be:

$$\sum_{j=1}^{w} g(x_{i_j})^{-2} \sum_{j=1}^{w} x_{i_j}^{2t-1} = 0. \tag{8.7}$$

We observe that the equations written thanks to this method are more complex than with Patterson algorithm, at least for two reasons. First, they are not directly polynomial, and the degrees implied are much higher. Second, as both $\mathbf{x}$ and g have to be unknown ([OS09][p. 125]), additive unknowns are necessary: either $t + 1$ to describe the secret polynomial's coefficients, or n if we introduce new equations $y_i = g(x_i)^{-2}$. We conclude that the alternant decoder is intrinsically more resistant to Strenzke's attacks. However, the overall security is still not clear due to the uncertainty on the counter-measure (described in Algorithm 11) against Algorithm 9.

8.2 Extended euclidean Algorithm with constant flow

We expose a way of implementing the EEA algorithm unused so far for McEliece decryption. It has the very interesting property of requiring a number of operations depending only on the Goppa polynomial degree t and *not* on the weight of the error introduced in the ciphertext. Therefore, the attacks of 8.1.1 and 8.1.2 are not possible.

It is inspired by Berlekamp's work in [BSP94] (which as followed by other works of optimization in the VLSI community, amongst many others [SS01a, SY09]). We could find no reference to it in any paper related to McEliece. On the contrary, designing such an algorithm is desirable goal according to the conclusion of [Str13]. The reason may be that [BSP94] has a very limited access, and we could find no completeness proofs of the algorithm proposed. Here, we transform smoothly the original EEA (Algorithm 1) into successive versions gaining in regularity (Algorithms 14 and 15). We end up with Algorithm 16, which is simpler and more regular than all the previous ones. At each step, we give and prove in Section 8.3 the

form of the outputs and intermediate values. Finally, each execution of Algorithm 16 costs, in field multiplications, exactly $16t^2$ ($2t$ times a loop costing $4 \times 2t$).

In the rest of this article we will set N be the number of Euclidean divisions performed during $\mathrm{EEA}(z^{2t}, S_{Alt}(z), t)$ in Algorithm 1, $d_i = \deg(r_i(z))$, and $\delta_i = \deg(q_i(z)) = \deg(r_{i-2}) - \deg(r_{i-1})$.

For any polynomial $P(z) \in \mathbb{F}_{q^m}[z]$, we denote its coefficients by P_j even for $j > \deg(P)$ (in which case $P_j = 0$), so that

$$P(z) = \sum_{j=0}^{+\infty} P_j z^j = P_{\deg(P)} z^{\deg(P)} + \cdots + P_0.$$

Regarding the δ_i's, we prove the following result which will be very useful to design an algorithm with regular pattern:

Lemma 8.4. *Let the δ_i's be defined as previously during the execution of* $\mathrm{EEA}(z^{2t}, S_{\mathbf{e}}(z), t)$, *then it holds that*

$$\sum_{i=1}^{N} \delta_i = \deg(u_N(z)) = \deg(\omega_{inv,\mathbf{e}}) = w_H(\mathbf{e}) - 1.$$

We propose several intermediate versions of algorithms computing $\sigma_{inv,\mathbf{e}}(z)$.

8.2.1 Unrolling Euclidean divisions.

In Algorithm 14 (on page 152), we decompose each Euclidean division into a number of polynomial subtractions depending only on δ_i the degrees of the quotients. We give explicit forms of the intermediate values of the Euclidean division of $R_{i-2}(z)$ by $R_{i-1}(z)$, that we denote by $R_i^{(0)}(z), \ldots, R_i^{(\delta_i+1)}(z)$. To do so, we eliminate in each $R_i^{(j)}(z)$ (for $0 \leqslant j \leqslant \delta_i + 1$) the term $z^{d_{i-2}-j}$, whether the associated coefficient is zero or not. This is why we perform the Euclidean divisions so as to avoid the divisions by a field elements (Steps 7 to 11 of Algorithm 14). Consequently, the outputs are multiple of the outputs of Algorithm 1 with the same inputs.

Proposition 8.5 (Comparison of Algorithms 1 and 14)**.** *Let $a(z)$ and $b(z)$ be two polynomials with $\deg(a(z)) \geqslant \deg(b(z))$, and d a non-negative integer. $u_i(z), v_i(z), r_i(z), q_i(z)$ are the intermediate values in Algorithm 1, and $U_i(z), V_i(z), R_i(z)$ are the intermediate values in Algorithm 14. It holds that, for all $i = -1, \ldots, N$, there exists $\lambda_i \in \mathbb{F}_{q^m}^*$ such that:*

$$\begin{aligned} R_i(z) &= \lambda_i r_i(z), \\ U_i(z) &= \lambda_i u_i(z). \end{aligned}$$

As a consequence, $\Delta_i = \deg(R_{i-2}) - \deg(R_{i-1}) = \deg(r_{i-2}) - \deg(r_{i-1}) = \delta_i$ for all i.

There are two problems with Algorithm 14: the first one is that the inner **for** loop ((Steps 7 to 11) has a variable length, and contains a multiplication $z^{\delta_i-(j-1)}R_i(z)$ which depends on the iteration, which will produce a recognizable pattern. The second problem is that the **while** loop leads to a variable number of operations according to the input. Algorithm 15 is a first step towards the resolution of the second problem. It is not realistic (it requires that Algorithm 14 has already been executed and observed), but it eases the proofs of completeness of Algorithm 16, which solves both issues.

8.2.2 Regular polynomial shift pattern.

In Algorithm 15 (on page 153), we perform the Euclidean division in such a way that we only multiply the operand by z at each **for** iteration. This can be done by splitting in two phases each Euclidean divisions. The first phase (Steps 4 to 7) "re-aligns" the operands $\tilde{R}_{i-2}$ and $\tilde{R}_{i-1}$ so that they both have same degree $d = \deg(R_{-1}(z))(= 2t)$. Doing so, the second phase (Steps 8 to 12) compute the polynomial subtractions (corresponding to Steps 9-10 of Algorithm 14) and perform a shift "re-aligning" the operands. A consequence is that the polynomials $\tilde{R}_i(z)$ are of the form $z^{k_i}R_i(z)$ and the degrees d_i are lost. This problem will be solved in Algorithm 16.

Proposition 8.6 (Comparison of Algorithms 14 and 15)**.** *For each $i = 1, \ldots, N$, after Step 13 of Algorithm 15, it holds that*

$$\begin{aligned}(\tilde{R}_{i-1}(z), \tilde{R}_i(z)) &= (z^{d-d_{i-1}}R_{i-1}(z), z^{d-d_{i-1}+1}R_i(z)),\\ (\tilde{U}_{i-1}(z), \tilde{U}_i(z)) &= (z^{d-d_{i-1}}U_{i-1}(z), z^{d-d_{i-1}+1}U_i(z)).\end{aligned}$$

8.2.3 Complete Regular Flow EEA.

To design a real constant flow algorithm, we merge the loops L_1 and L_2 in a common pattern so as to be indistinguishable (Steps 5 to 7 of Algorithm 16). They differenciate by the assignements which are performed in Steps 14-15 and 18-19. To know when polynomial substractions have to be stopped, we collect in a counter δ the number of shifts necessary to re-align the operands. Finally, when the polynomials σ_{inv} and ω_{inv} have been computed, the extra executions of the main loop (Steps 4 to 22) consist in shifting the operands. Therefore, the number of iterations can be safely set to the maximum value (*i.e.* $2t$ to decode the errors with $w_H(\mathbf{e}) = t$)), and the **while** loop is replaced by **for**.

Proposition 8.7 (Comparison of Algorithms 14 and 16.)**.** *For each $i = 1, \ldots, N$, after Step 21, it holds that:*

$$\begin{aligned}\hat{R}_{2(\delta_1+\cdots+\delta_i)}(z) &= z^{d-d_{i-1}+1}R_i(z),\\ \hat{U}_{2(\delta_1+\cdots+\delta_i)}(z) &= z^{d-d_{i-1}+1}U_i(z).\end{aligned}$$

The outputs of Algorithm 16 are, for some $\mu \in \mathbb{F}_{q^m}^*$:

$$\begin{aligned}\hat{R}_d(z) &= z^{d-w_H(\mathbf{e})+1}R_N(z) &&= \mu z^{d-w_H(\mathbf{e})+1}\omega_{inv}(z),\\ \hat{U}_d(z) &= z^{d-w_H(\mathbf{e})+1}U_N(z) &&= \mu z^{d-w_H(\mathbf{e})+1}\sigma_{inv}(z).\end{aligned}$$

Therefore, provided 0 is not an element of $\mathbf{x}$, $\hat{U}_d(z)$ allows to recover the error positions without ambiguity. Transposing this result to Patterson decoding requires to adapt both EEA's. The adaptation of the second one is straightforward. For the first one (syndrome inversion), a problem arises: the analogous of Proposition 8.7 would yield $\hat{U}_{N_I}(z) = \mu z^{k_i}(S^{-1}_{Gop,\mathbf{e}} \bmod g)$ for some $k_i > 0$, and we found no way of determining when z is a factor of $S^{-1}_{Gop,\mathbf{e}} \bmod g$. However, we can protect the second EEA to avoid the attack of Paragraph 8.1.2.

Conclusion.

We proposed an algorithm determining the error-locator polynomial costing always $16t^2$ field multiplications on any input. It contains a test depending on secret data, followed by two balanced branches. The indistinguishability of those branches by an attacker is crucial for the security of the decryption, and depends on the architecture of the implementation. This is

Algorithm 16 EEA with regular flow

Input: $a(z) = z^{2t}, b(z) = S_{\mathbf{e}}(z), d = 2t$

Output: $\hat{U}_d(z) = \mu z^{d-w_H(\mathbf{e})+1}\sigma_{inv}(z)$,$\hat{R}_d(z) = \mu z^{d-w_H(\mathbf{e})+1}\omega_{\mathbf{e}}(z)$ for some $\mu \in \mathbb{F}_{q^m}^*$.

1: $\hat{R}_{-1}(z) \leftarrow a(z), \hat{R}_0(z) \leftarrow zb(z)$,
2: $\hat{U}_{-1}(z) \leftarrow 1, \hat{U}_0(z) \leftarrow 0$,
3: $\delta \leftarrow -1$.
4: **for** $j = 1, \ldots, d$ **do**
5: $\quad \alpha_j \leftarrow \hat{R}_{j-1,d}, \beta_j \leftarrow \hat{R}_{j-2,d}$.
6: $\quad temp_R(z) \leftarrow z\left(\alpha_j \hat{R}_{j-2}(z) - \beta_j \hat{R}_{j-1}(z)\right)$.
7: $\quad temp_U(z) \leftarrow z\left(\alpha_j \hat{U}_{j-2}(z) - \beta_j \hat{U}_{j-1}(z)\right)$.
8: $\quad$ **if** $\alpha_j = 0$ (ie $\deg(\hat{R}_{j-1}) < \deg(\hat{R}_{j-2})$) **then**
9: $\quad\quad \delta \leftarrow \delta + 1$.
10: $\quad$ **else**
11: $\quad\quad \delta \leftarrow \delta - 1$.
12: $\quad$ **end if**
13: $\quad$ **if** $\delta < 0$ **then**
14: $\quad\quad (\hat{R}_j(z), \hat{R}_{j-1}(z)) \leftarrow (\hat{R}_{j-1}(z), temp_R)$
15: $\quad\quad (\hat{U}_j(z), \hat{U}_{j-1}(z)) \leftarrow (\hat{U}_{j-1}(z), temp_U)$
16: $\quad\quad \delta \leftarrow 0$.
17: $\quad$ **else**
18: $\quad\quad (\hat{R}_j(z), \hat{R}_{j-1}(z)) \leftarrow (temp_R, \hat{R}_{j-2}(z))$
19: $\quad\quad (\hat{U}_j(z), \hat{U}_{j-1}(z)) \leftarrow (temp_U, \hat{U}_{j-2}(z))$
20: $\quad\quad \delta \leftarrow \delta$.
21: $\quad$ **end if**
22: **end for** $\quad L$
23: **return** $\hat{U}_d(z), \hat{R}_d(z)$

only a part of the decryption, but the other parts (matrix-vector products, root-finding) had received more attention. The next step is to give a complete implementation, and evaluate its resistance to other leakages than timing. Moreover, the resistance to fault injection has not been treated here.

Algorithm 14 EEA (unrolled Euclidean Division)

Input: $a(z) = z^{2t}, b(z) = S_{\mathbf{e}}(z)$.
Output: $U(z) = \lambda_N \sigma_{\mathbf{e}}(z), R(z) = \lambda_N \omega_{\mathbf{e}}(z)$ (for some $\lambda_N \in \mathbb{F}_{q^m}^*$).

$(R_{-1}(z), R_0(z)) \leftarrow (a(z), b(z))$,$(U_{-1}(z), U_0(z)) \leftarrow (1, 0)$, $i \leftarrow 0$.
while $\deg(R_i(z)) > t$ **do**
  $i \leftarrow i + 1$
  $R_{i-2}^{(0)}(z) \leftarrow R_{i-2}(z)$,$U_{i-2}^{(0)}(z) \leftarrow U_{i-2}(z)$
  $\Delta_i \leftarrow \deg(R_{i-2}) - \deg(R_{i-1})$
  $\beta_i \leftarrow \mathrm{LC}(R_{i-1}(z))$.
  for $j = 0, \ldots, \Delta_i$ **do**
    $\alpha_{i,j} \leftarrow R_{i,d_{i-2}-j}^{(j)}$,
    $R_{i-2}^{(j+1)}(z) \leftarrow \beta_i R_{i-2}^{(j)}(z) - \alpha_{i,j} z^{\Delta_i - j} R_{i-1}(z)$
    $U_{i-2}^{(j+1)}(z) \leftarrow \beta_i U_{i-2}^{(j)}(z) - \alpha_{i,j} z^{\Delta_i - j} U_{i-1}(z)$
  end for
  $R_i(z) \leftarrow R_{i-2}^{(\Delta_i+1)}(z)$,$U_i(z) \leftarrow U_{i-2}^{(\Delta_i+1)}(z)$
end while
$N \leftarrow i$.
return $U_N(z), R_N(z)$

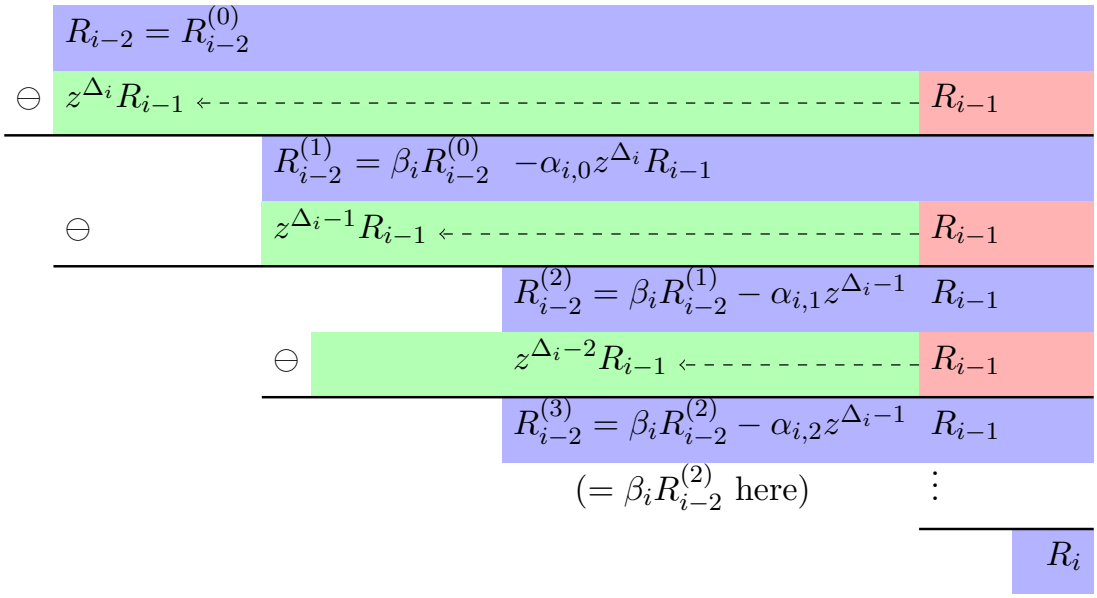


Figure 8.3 – Euclidean division of R_{i-2} by R_{i-1} in exactly $\Delta_i + 1$ polynomial subtractions (with $\Delta_i = \deg(R_{i-2}) - \deg(R_{i-1})$). In this example, $\alpha_{i,2} = 0$.

Algorithm 15 Toy EEA with regular shift pattern

Input: $a(z) = z^{2t}, b(z) = S_{\mathbf{e}}(z), d = 2t$
Output: $\tilde{U}_N(z) = z^{d-d_{N-1}+1}\sigma_{\mathbf{e}}(z), \tilde{R}_N(z) = z^{d-d_{N-1}+1}\omega_{\mathbf{e}}(z)$.

$\tilde{R}_{-1}(z) \leftarrow a(z), \tilde{R}_0(z) \leftarrow zb(z), \tilde{U}_{-1}(z) \leftarrow 1, \tilde{U}_0(z) \leftarrow 0.$
for $i = 1, \ldots, N$ **do**
  $\tilde{R}^{(0)}_{i-2}(z) \leftarrow \tilde{R}_{i-2}(z), \tilde{U}^{(0)}_{i-2}(z) \leftarrow \tilde{U}_{i-2}(z)$
  for $j = 1, \ldots, \Delta_i - 1$ **do** $\Big\}$ L_1
    $\tilde{R}_{i-1}(z) \leftarrow z\tilde{R}_{i-1}(z)$
    $\tilde{U}_{i-1}(z) \leftarrow z\tilde{U}_{i-1}(z)$
  end for
  for $j = 0, \ldots, \Delta_i$ **do** $\Big\}$ L_2
    $\tilde{\alpha}_{i,j} \leftarrow \tilde{R}^{(j)}_{i,d}$, $\tilde{\beta}_i \leftarrow \tilde{R}_{i-1,d}$.
    $\tilde{R}^{(j+1)}_{i-2}(z) \leftarrow z\left(\tilde{\beta}_i\tilde{R}^{(j)}_{i-2}(z) - \tilde{\alpha}_{i,j}\tilde{R}_{i-1}(z)\right)$
    $\tilde{U}^{(j+1)}_{i-2}(z) \leftarrow z\left(\tilde{\beta}_i\tilde{U}^{(j)}_{i-2}(z) - \tilde{\alpha}_{i,j}\tilde{U}_{i-1}(z)\right)$
  end for
  $\tilde{R}_i(z) \leftarrow \tilde{R}^{(\Delta_i+1)}_{i-2}(z), \tilde{U}_i(z) \leftarrow \tilde{U}^{(\Delta_i+1)}_{i-2}(z)$
end for
return $\tilde{U}_N(z), \tilde{R}_N(z)$

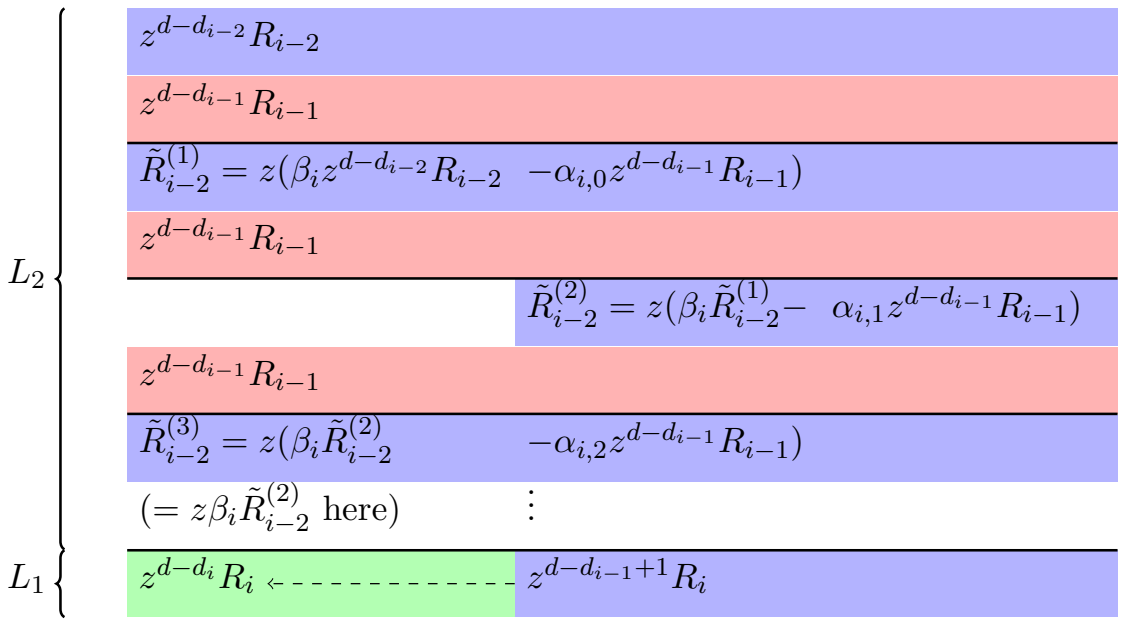


Figure 8.4 – "Re-aligned" Euclidean division of R_{i-2} by R_{i-1}: inputs and outputs are degree$-d$ polynomials $z^{d-d_{i-2}}R_{i-2}$, $z^{d-d_{i-1}}R_{i-1}$, and $z^{d-d_i}R_i$.

8.3 Proofs of completeness

We give the proofs of the results of Section 8.2. It consists in comparing the outputs and intermediate results of Algorithms 14, 15, and 16.

8.3.1 Proof of Proposition 8.5.

We recall of the $R_i(z)$'s are computed: set $R_{-1}(z) = r_{-1}(z) = a(z)$ and $R_0(z) = r_0(z) = b(z)$. Then, for $1 \leqslant i \leqslant N$, set $R_{i-2}^{(0)}(z) = R_{i-2}(z)$ and recursively

$$R_{i-2}^{(j+1)}(z) = \beta_i R_{i-2}^{(j)}(z) - \alpha_{i,j} z^{\delta_i - j} R_{i-1}(z) \tag{8.8}$$

with $\alpha_{i,j} = R_{i,d_{i-2}-j}^{(j)}$, $\beta_i = \mathrm{LC}(R_{i-1}(z))$. $R_i(z)$ is defined by $R_i(z) = R_{i-2}^{(\delta_i+1)}(z)$.

Remark 8.8. *The notation $R^{(j)}(z)$ should not be mistaken with the Fröbenius action defined in Chapter 7.*

In essence we have to prove that $R_{i-2}^{(\delta_i+1)}(z)$ is the rest of the Euclidian division of $R_{i-2}(z)$ by $R_{i-1}(z)$. First, we prove by induction on j that, for all $1 \leqslant i \leqslant N$ and $0 \leqslant j \leqslant \delta_i + 1$, there exists a polynomial $f_{i,j}(z) \in \mathbb{F}_{q^m}[z]$ and a non-zero scalar $\mu_{i,j}$ such that

$$R_{i-2}^{(j)}(z) = \mu_{i,j} R_{i-2}^{(0)}(z) - f_{i,j}(z) R_{i-1}(z) \text{ and } \deg(R_{i-2}^{(j)}(z)) \leqslant d_{i-2} - j. \tag{8.9}$$

For $j = 0$, pick $f_{i,0}(z) = 0, \mu_{i,0} = 1$ and notice that $\deg(R_{i-2}^{(0)}) = d_{i-2}$. Now at step $j > 0$, suppose $R_{i-2}^{(j)}(z) = \mu_{i,j} R_{i-2}^{(0)}(z) - f_{i,j}(z) R_{i-1}(z)$, then

$$R_{i-2}^{(j+1)}(z) = \beta_i R_{i-2}^{(j)}(z) - \alpha_{i,j} z^{\delta_i - j} R_{i-1}(z) = \underbrace{\mu_{i,j}\beta_i}_{\mu_{i,j+1}} R_{i-2}^{(0)}(z) - \underbrace{(f_{i,j}(z) + \alpha_{i,j} z^{\delta_i - j})}_{f_{i,j+1}(z)} R_{i-1}(z).$$

Regarding the degree, if $\deg(R_{i-2}^{(j)}) < d_{i-2} - j$, then $R_{i-2,d_{i-2}-j}^{(j)} = 0$, so $\alpha_{i,j} = 0$ and $\deg(R_{i-2}^{(j+1)}) = \deg(R_{i-2}^{(j)}) \leqslant d_{i-2} - (j+1)$. If $\deg(R_{i-2}^{(j)}) = d_{i-2} - j$, then observe that $LC(\beta_i R_{i-2}^{(j)}) = LC(R_{i-1}(z)) R_{i,d_{i-2}-j}$ and $LC(\alpha_{i,j} z^{\delta_i - j} R_{i-1}(z)) = R_{i,d_{i-2}-j} LC(R_{i-1}(z))$, so that the leading monomials cancel, and therefore $\deg(R_{i-2}^{(j+1)}) \leqslant d_{i-2} - (j+1)$. Thanks to Eq. (8.9), we know that at step $j = \delta_i + 1$:

$$R_{i-2}(z) = f_{i,\delta_i+1}(z) R_{i-1}(z) + \mu_{i,\delta_i+1} R_{i-2}^{(\delta_i+1)}(z) \text{ and } \deg(R_{i-2}^{(\delta_i+1)}) \leqslant d_{i-1} - 1.$$

Thanks to the induction hypothesis $R_{i-2}(z) = \lambda_{i-2} r_{i-2}(z)$ and $R_{i-1}(z) = \lambda_{i-1} r_{i-1}(z)$, so we have

$$r_{i-2}(z) = \lambda_{i-1}\lambda_{i-2}^{-1} f_{i,\delta_i+1}(z) r_{i-1}(z) + \lambda_{i-2}^{-1}\mu_{i,\delta_i+1} R_{i-2}^{(\delta_i+1)}(z).$$

By unicity in the Euclidean division, we set $\lambda_i = \lambda_{i-2}\mu_{i,\delta_i+1}^{-1}$ it follows that:

$$R_{i-2}^{(\delta_i+1)}(z) \quad = \quad \lambda_i r_i(z), \qquad \lambda_{i-1}\lambda_{i-2}^{-1} f_{i,d_{i-2}-d_i+1}(z) = q_i(z). \tag{8.10}$$

For $(U_{i-2}^{(j)}(z))$, we would prove the same way that $U_{i-2}^{(j)}(z) = \mu_{i,j}U_{i-2}^{(0)}(z) - f_{i,j}(z)U_{i-1}(z)$ with the same $\mu_{i,j}$ and $f_{i,j}(z)$. Therefore, thanks to the induction hypothesis ($U_{i-2}(z) = \lambda_{i-2}u_{i-2}(z)$ and $U_{i-1}(z) = \lambda_{i-1}u_{i-1}(z)$),

$$u_{i-2}(z) = \underbrace{\lambda_{i-1}\lambda_{i-2}^{-1}f_{i,\delta_i+1}(z)}_{q_i(z)(8.10)}\, u_{i-1}(z) + \underbrace{\lambda_{i-2}^{-1}\mu_{i,\delta_i+1}}_{\lambda_i^{-1}(8.10)}\, U_{i-2}^{(\delta_i+1)}(z).$$

With Eq. (8.10), we see that $U_{i-2}^{(\delta_i+1)}(z) = \lambda_i u_{i-2}(z) - \lambda_i q_i(z)u_{i-1}(z) = \lambda_i u_i(z)$. □

8.3.2 Proof of Proposition 8.6.

The operands at entering the main **for** loop (Steps 2-14 of Alg. 15) are by induction hypothesis $(\tilde{R}_{i-2}, \tilde{R}_{i-1}) = z^{d-d_{i-2}}R_{i-2}(z), z^{d-d_{i-2}+1}R_{i-1}(z))$(for $i = 1$, we have indeed $(\tilde{R}_{-1}, \tilde{R}_0) = (R_{-1}(z), zR_0(z))$).

The main loop is decomposed into two phases: L_1 (Steps 4-7) and L_2 (Steps 8-12). L_1 consists in shifting $z^{d-d_{i-2}+1}R_i(z)$ up-wise until it has a non-zero d-th coefficient (*ie* it has degree d), that is exactly $\Delta_i - 1 = d_{i-2} - d_{i-1} - 1$ times, so that after Step 7 $\tilde{R}_{i-1}(z) = z^{d-d_{i-1}}R_{i-1}(z)$.

L_2 (Steps 8-12) performs a twisted Euclidean division of $R_{i-2}(z)$ by $R_{i-1}(z)$ on the shifted values $\tilde{R}_{i-2}(z)z^{d-d_{i-2}}R_{i-2}(z)$ and $\tilde{R}_{i-1}(z) = z^{d-d_{i-1}}R_{i-1}(z)$. We prove that, after Step 12, $\tilde{R}_{i-2}^{(\delta_i+1)}(z) = z^{d-d_{i-1}+1}R_i(z)$. To do so, we show the following link between the intermediate results of Algorithms 14 ($R_{i-2}^{(j)}(z)$) and 15 ($\tilde{R}_{i-2}^{(j)}(z)$): for $0 \leqslant j \leqslant \Delta_i + 1$,

$$\tilde{R}_{i-2}^{(j)}(z) = z^{d-d_{i-2}+j}R_{i-2}^{(j)}(z). \tag{8.11}$$

This is done by induction on j. For $j = 0$, we have $\tilde{R}_{i-2}^{(0)}(z) = z^{d-d_{i-2}}R_{i-2}(z) = z^{d-d_{i-2}}R_{i-2}^{(0)}(z)$. Then, suppose $\tilde{R}_{i-2}^{(j)}(z) = z^{d-d_i+j}R_{i-2}^{(j)}(z)$. So, it holds that $\tilde{R}_{i-2,d}^{(j)} = R_{i-2,d-d_{i-2}+j}^{(j)}$, and $\mathrm{LC}(\tilde{R}_{i-1}(z)) = \mathrm{LC}(R_{i-1}(z))$, so that $\tilde{\alpha}_{i,j} = \alpha_{i,j}$ and $\tilde{\beta}_i = \beta_i$. Now $\tilde{R}_{i-2}^{(j+1)}(z)$ is given by:

$$\begin{aligned}\tilde{R}_{i-2}^{(j+1)}(z) &= z\left(\tilde{\beta}_i\tilde{R}_{i-2}^{(j)}(z) - \tilde{\alpha}_{i,j}\tilde{R}_{i-1}(z)\right) = z\left(z^{d-d_{i-2}+j}\beta_i R_{i-2}^{(j)}(z) - \alpha_{i,j}z^{d-d_{i-1}}R_{i-1}(z)\right)\\ &= z^{d-d_{i-2}+j}\left(\beta_i R_{i-2}^{(j)}(z) - \alpha_{i,j}z^{\delta_i-(j)}\tilde{R}_{i-1}(z)\right) \underset{\text{Eq (8.8)}}{=} z^{d-d_{i-2}+j}R_{i-2}^{(j)}(z).\end{aligned}$$

So (8.11) is proved by induction. With $j = \delta_i+1$, we obtain $\tilde{R}_{i-2}^{(\delta_i+1)}(z) = z^{d-d_{i-1}+1}R_{i-2}^{(\delta_i+1)}(z) = z^{d-d_{i-1}+1}R_i(z)$ (thanks to Eq. (8.10)): the announced result holds for $\tilde{R}_i(z)$. The proof is exactly the same for $\tilde{U}_i$ by substituting R by U in the previous proof.

8.3.3 Proof of Proposition 8.7.

By induction, we suppose that $\hat{R}_{2(\delta_1+\cdots+\delta_{i-1})}(z) = z^{d-d_{i-2}+1}R_{i-1}(z)$. As in the proof of Proposition 8.6, we distinguish several phases when iterating the main **for** loop L (Steps 4-22). First, for $j = 0, \ldots, \delta_i - 1$, $\hat{R}_{2(\delta_1+\cdots+\delta_{i-1})+j} = z^{d-d_{i-2}+1}R_{i-1}(z)$ has degree lower than d and L consists in shifting $\hat{R}$ up-wise (up to a multiplicative constant) as L_1 in Alg 15. In addition, we update the counter δ to record the degree difference between R_{i-2} and R_{i-1}. Indeed, this phase stops when $z^{d-d_{i-2}+1+j}R_{i-1}(z)$ reaches degree d, that is when $j = d_{i-2}-d_{i-1}-1 = \delta_i-1$. Second, for $j = \delta_i, \ldots, 2\delta_i - 1$, L processes the Euclidean division of $R_{i-2}(z)$ by $R_{i-1}(z)$ as L_2 in Alg 15. The intermediate results are exactly those detailed in Proposition 8.6. The

difference here is that, prior to computing the last step of the division, δ is equal to -1, so that the intermediate values are swapped, and the announced result for $\hat{R}_{2(\delta_1+\cdots+\delta_i)}(z)$ holds.

We deduce that $\hat{R}_{2(\delta_1+\cdots+\delta_N)}(z) = z^{d-d_{N-1}+1}R_N(z) = \lambda_N z^{d-d_{N-1}+1}\sigma_{inv}(z)$ and the same proof shows that $\hat{U}_{2(\delta_1+\cdots+\delta_N)}(z) = z^{d-d_{N-1}+1}U_N(z) = \lambda_N z^{d-d_{N-1}+1}\omega_{inv}(z)$. The following iterations ($j > 2w_H(\mathbf{e})$) consists only in shifts. Indeed, as $\deg(z^{d-d_{N-1}+1}R_N(z)) = w_H(\mathbf{e}) + 1 + \deg(\omega_{\mathbf{e}}(z)) \leqslant 2w_H(\mathbf{e})$, and after $2t - 2w_H(\mathbf{e})$ up-wise shifts the output still has degree $\leqslant d = 2t$. $\square$

Bibliography

[AHPT11] Roberto Avanzi, Simon Hoerder, Dan Page, and Michael Tunstall. Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. *J. Cryptographic Engineering*, 1(4):271–281, 2011. 8, 22, 137, 139, 140

[Bar04] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Université de Paris VI, 2004. 65, 66

[BB11] Morgan Barbier and Paulo S. L. M. Barreto. Key reduction of McEliece's cryptosystem using list decoding. In Alexander Kuleshov, Vladimir Blinovsky, and Anthony Ephremides, editors, *2011 IEEE International Symposium on Information Theory Proceedings, ISIT 2011, St. Petersburg, Russia, July 31 - August 5, 2011*, pages 2681–2685. IEEE, 2011. 73

[BBC+14] Marco Baldi, Marco Bianchi, Franco Chiaraluce, Joachim Rosenthal, and Davide Schipani. Enhanced Public Key Security for the McEliece Cryptosystem. *Journal of Cryptology*, pages 1–27, 2014. 55

[BCGO09] T. P. Berger, P.L. Cayrel, P. Gaborit, and A. Otmani. Reducing Key Length of the McEliece Cryptosystem. In Bart Preneel, editor, *Progress in Cryptology - Second International Conference on Cryptology in Africa (AFRICACRYPT 2009)*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97, Gammarth, Tunisia, June 21-25 2009. iv, 6, 9, 10, 20, 21, 23, 24, 81, 82, 83, 88, 89, 93

[BCMN10] Paulo S. L. M. Barreto, Pierre-Louis Cayrel, Rafael Misoczki, and Robert Niebuhr. Quasi-Dyadic CFS Signatures. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 336–349. Springer, 2010. iv, 6, 7, 9, 10, 20, 21, 22, 23, 24, 25, 73, 81, 82, 83, 88, 89, 93, 105, 110, 111, 113, 114

[BCP97a] W. Bosma, J. J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *J. Symb. Comput.*, 24(3/4):235–265, 1997. 12, 27

[BCP97b] Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma algebra system I: The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997. 110, 113, 133

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast Constant-Time Code-Based Cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272. Springer, 2013. 8, 22, 45

[Ber] Elwyn R. Berlekamp. *Algebraic coding theory.* McGraw-Hill series in systems science. McGraw-Hill, [cop. 1968], New York. 39

[Ber97] Thomas A. Berson. Failure of the McEliece Public-Key Cryptosystem Under Message-Resend and Related-Message Attack. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 213–220. Springer, 1997. 8, 23

[Ber99] T. P. Berger. Cyclic alternant codes induced by an automorphism of a GRS code. In R. Mullin and G. Mullen, editors, *Finite fields: Theory, Applications and Algorithms*, volume 225, pages 143–154, Waterloo, Canada, 1999. AMS, Contemporary Mathematics. 14, 15, 29, 30, 81, 84, 88, 89

[Ber00a] T. P. Berger. On the cyclicity of Goppa codes, parity-check subcodes of Goppa codes and extended Goppa codes. *Finite Fields and Applications*, 6:255–281, 2000. 39, 84

[Ber00b] Thierry P. Berger. Goppa and related codes invariant under a prescribed permutation. *IEEE Transactions on Information Theory*, 46(7):2628–2633, 2000. 14, 29, 84, 88

[Ber10] Daniel J. Bernstein. Grover vs. Mceliece. In *Proceedings of the Third International Conference on Post-Quantum Cryptography*, PQCrypto'10, pages 73–80, Berlin, Heidelberg, 2010. Springer-Verlag. 49

[BFP11] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Cryptanalysis of Multivariate and Odd-Characteristic HFE Variants. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2011. 7, 21

[BFS15] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the Gröbner basis algorithm. *Journal of Symbolic Computation*, 70(0):49 – 70, 2015. 63

[Bis10] Bhaskar Biswas. *Aspects de mise en oeuvre de la cryptographie basée sur les codes.* Thèse, École Polytechnique X, October 2010. 8, 22, 45, 137

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. 49

[BL05] T. P. Berger and P. Loidreau. How to Mask the Structure of Codes for a Cryptographic Use. *Designs Codes and Cryptography*, 35(1):63–79, 2005. 6, 20, 50

[BL11] Andrej Bogdanov and Chin Ho Lee. Homomorphic encryption from codes. *CoRR*, abs/1111.4301, 2011. 55

[BLM11] Paulo S. L. M. Barreto, Richard Lindner, and Rafael Misoczki. Monoidic Codes in Cryptography. In Yang [Yan11], pages 179–199. iv, 6, 7, 9, 10, 20, 21, 22, 23, 24, 25, 81, 82, 83, 84, 88, 89, 91, 93, 105, 110, 111, 113, 114, 115

[BLP08] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and Defending the McEliece Cryptosystem. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography, Second International Workshop, PQCrypto*

2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008. 49

[BLP11a] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller Decoding Exponents: Ball-Collision Decoding. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. 49

[BLP11b] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Wild McEliece. In *Proceedings of the 17th international conference on Selected areas in cryptography*, SAC'10, pages 143–158, Berlin, Heidelberg, 2011. Springer-Verlag. 6, 10, 12, 20, 21, 25, 26, 117, 125, 126, 130, 133, 134

[BLP11c] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Wild Mceliece Incognito. In *Proceedings of the 4th International Conference on Post-Quantum Cryptography*, PQCrypto'11, pages 244–254, Berlin, Heidelberg, 2011. Springer-Verlag. 6, 10, 12, 20, 21, 25, 26, 117, 125, 126, 130, 134

[BMvT78] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *Information Theory, IEEE Transactions on*, 24(3):384–386, May 1978. 4, 18, 49

[BSP94] Elwyn Berlekamp, Gadiel Seroussi, and Tong Po. A hypersystolic Reed–Solomon decoder. In Vijay K. Bhargava, Stephen B. Wicker, IEEE Communications Society, and IEEE Information Theory Society, editors, *Reed-Solomon codes and their applications*, pages 205–241. IEEE Press Piscataway, NJ, 1994. 137, 148

[Buc65] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Innsbruck, 1965. 7, 57, 60, 61

[CC98] A. Canteaut and F. Chabaud. A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. 14, 29, 49

[CFS01] N. T. Courtois, M. Finiasz, and N. Sendrier. How to Achieve a McEliece-Based Digital Signature Scheme. *Lecture Notes in Computer Science*, 2248:157–174, 2001. 5, 19

[CGG+14] Alain Couvreur, Philippe Gaborit, Valérie Gauthier-Umaña, Ayoub Otmani, and Jean-Pierre Tillich. Distinguisher-based attacks on public-key cryptosystems using Reed-Solomon codes. *Designs, Codes and Cryptography*, 73(2):641–666, 2014. 14, 30, 55

[CGP08] Pierre-Louis Cayrel, Philippe Gaborit, and Emmanuel Prouff. Secure Implementation of the Stern Authentication and Signature Schemes for Low-Resource Devices. In Gilles Grimaud and François-Xavier Standaert, editors, *CARDIS*, volume 5189 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2008. 8, 22

[CLO07] D. A. Cox, J. B. Little, and D. O'Shea. *Ideals, Varieties, and algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra.* Undergraduate Texts in Mathematics, Springer-Verlag, New York., 2007. 57

[CM03] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003. 7, 21

[CMCP14] Alain Couvreur, Irene Marquez-Corbella, and Ruud Pellikaan. A polynomial time attack against algebraic geometry code based public key cryptosystems. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014* [DBL14], pages 1446–1450. 6, 20, 50, 55

[COT14] Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. Polynomial Time Attack on Wild McEliece over Quadratic Extensions. In *EUROCRYPT*, page to appear, 2014. 15, 30, 55, 134

[COTGU15] Alain Couvreur, Ayoub Otmani, Jean-Pierre Tillich, and Valérie Gauthier-Umaña. A Polynomial-Time Attack on the BBCRS Cryptosystem. In *Public-Key Cryptography - PKC*, page to appear, 2015. 55

[Dal07] Léonard Dallot. Towards a Concrete Security Proof of Courtois, Finiasz and Sendrier Signature Scheme. In Stefan Lucks, Ahmad-Reza Sadeghi, and Christopher Wolf, editors, *WEWoRC*, volume 4945 of *Lecture Notes in Computer Science*, pages 65–77. Springer, 2007. 55

[DBL14] *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*. IEEE, 2014. 160, 161

[DDMQN12] Nico Döttling, Rafael Dowsley, Jörn Müller-Quade, and Anderson C. A. Nascimento. A CCA2 Secure Variant of the McEliece Cryptosystem. *IEEE Transactions on Information Theory*, 58(10):6672–6680, 2012. 8, 23, 139

[Del75] P. Delsarte. On subfield subcodes of modified Reed-Solomon codes. *IEEE Transactions on Information Theory*, 21(5):575–576, 1975. 36

[DMR11] Hang Dinh, Cristopher Moore, and Alexander Russell. McEliece and Niederreiter Cryptosystems That Resist Quantum Fourier Sampling Attacks. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 761–779. Springer Berlin Heidelberg, 2011. 54

[DMS03] I. Dumer, D. Micciancio, and M. Sudan. Hardness of approximating the minimum distance of a linear code. *Information Theory, IEEE Transactions on*, 49(1):22–37, Jan 2003. 35

[Dür87] Arne Dür. The automorphism groups of Reed-Solomon codes. *J. Comb. Theory Ser. A*, 44(1):69–82, January 1987. 14, 15, 29, 30, 38, 81, 84, 85, 86

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4) . *Journal of Pure and Applied Algebra*, 139(1–3):61 – 88, 1999. 63, 110, 113, 133

[Fau02] Jean Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM. 63, 110

[Fau10] Jean-Charles Faugère. FGb: A Library for Computing Gröbner Bases. In Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 84–87, Berlin, Heidelberg, September 2010. Springer Berlin / Heidelberg. 63, 110

[FGHR14] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaël Renault. Sub-cubic change of ordering for Gröbner basis: a probabilistic approach. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó,

editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 170–177. ACM, 2014. 64

[FGLM93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering. *J. Symb. Comput.*, 16(4):329–344, 1993. 63, 64

[FGO+11] Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high rate mceliece cryptosystems. In *2011 IEEE Information Theory Workshop, ITW 2011, Paraty, Brazil, October 16-20, 2011*, pages 282–286. IEEE, 2011. 74

[FGO+13] Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate mceliece cryptosystems. *IEEE Transactions on Information Theory*, 59(10):6830–6844, 2013. 7, 21, 55, 74

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 44–60. Springer, 2003. 7, 21

[FLdVP08] J.-C. Faugère, F. Levy-dit Vehel, , and L. Perret. Cryptanalysis of Minrank. In David Wagner, editor, *Advances in Cryptology - CRYPTO'08*, volume 5157, pages 280–296, 2008. 7, 21

[FOP+14a] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric de Portzamparc, and Jean-Pierre Tillich. Folding Alternant and Goppa Codes with Non-Trivial Automorphism Groups. *IACR Cryptology ePrint Archive*, 2014:353, 2014. 81

[FOP+14b] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric de Portzamparc, and Jean-Pierre Tillich. Structural weakness of compact variants of the mceliece cryptosystem. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014* [DBL14], pages 1717–1721. 81

[FOP+15] J. C. Faugère, A. Otmani, L. Perret, F. de Portzamparc, and J. P. Tillich. Structural Cryptanalysis of McEliece-Like Schemes with Symmetric Keys, october 2015. *Designs, Codes and Cryptography.* 105

[FOPT10a] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic Cryptanalysis of McEliece Variants with Compact Keys. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer, 2010. 6, 7, 9, 10, 12, 20, 21, 25, 27, 54, 71, 105, 106, 110, 115, 120, 121

[FOPT10b] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic Cryptanalysis of McEliece Variants with Compact Keys – Toward a Complexity Analysis. In *SCC '10: Proceedings of the 2nd International Conference on Symbolic Computation and Cryptography*, pages 45–55, RHUL, June 2010. 66, 72, 110

[FPdP14] Jean-Charles Faugère, Ludovic Perret, and Frédéric de Portzamparc. Algebraic Attack against Variants of McEliece with Goppa Polynomial of a Special Form. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application*

of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I, volume 8873 of *Lecture Notes in Computer Science*, pages 21–41. Springer, 2014. 117

[FS10] Jean-Charles Faugère and Pierre-Jean Spaenlehauer. Algebraic Cryptanalysis of the PKC'2009 Algebraic Surface Cryptosystem. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2010. 7, 21

[FS11] Matthieu Finiasz and Nicolas Sendrier. Digital Signature Scheme Based on McEliece. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 342–343. Springer, 2011. 77

[FSS11] 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. *Journal Of Symbolic Computation*, 46(4):406–437, 2011. Available online 4 November 2010. 13, 28, 66

[Gab85] E. M. Gabidulin. Theory of codes with maximum rank distance. *Problems of Information Transmission (English translation of Problemy Peredachi Informatsii)*, 21(1), 1985. 6, 20, 50

[Gab05] P. Gaborit. Shorter keys for code based cryptography. In *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, pages 81–91, Bergen, Norway, March 2005. 6, 20

[Gab13] Philippe Gaborit, editor. *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*. Springer, 2013. 166, 167

[GdP15] Mariya Georgieva and Frédéric de Portzamparc. Toward a Secure Implementation of McEliece Decryption. In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, page to appear, 2015. 137

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 7, 21

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '01, pages 251–261, London, UK, UK, 2001. Springer-Verlag. 7, 22

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014. 7, 22

[Hey11] Stefan Heyse. Implementation of McEliece Based on Quasi-dyadic Goppa Codes for Embedded Devices. In Yang [Yan11], pages 143–162. 8, 22, 137

[JM96] H. Janwa and O. Moreno. McEliece Public Key Cryptosystems Using Algebraic-Geometric Codes. *Designs Codes and Cryptography*, 8(3):293–307, 1996. 6, 20

[KI01] Kazukuni Kobara and Hideki Imai. Semantically Secure McEliece Public-Key Cryptosystems-Conversions for McEliece PKC. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 19–35, London, UK, UK, 2001. Springer-Verlag. 8, 23, 139

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag. 7, 22

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin Heidelberg, 1996. 7, 22

[Laz83] Daniel Lazard. Gröbner-Bases, Gaussian Elimination and Resolution of Systems of Algebraic Equations. In *Proceedings of the European Computer Algebra Conference on Computer Algebra*, EUROCAL '83, pages 146–156, London, UK, UK, 1983. Springer-Verlag. 61, 62, 66

[LB88] P. J. Lee and E. F. Brickell. An Observation on the Security of McEliece's Public-Key Cryptosystem. In *Advances in Cryptology - EUROCRYPT'88*, volume 330/1988 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988. 14, 29, 49

[LDW94] Yuan Xing Li, R.H. Deng, and Xin Mei Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *Information Theory, IEEE Transactions on*, 40(1):271–273, Jan 1994. 4, 18

[Leg11] Matthieu Legeay. Towards an approach using algebraic properties of the σ-subcode. In ed. by D. Augot and A. Canteaut, editors, *Proceedings of the Workshop on Coding and Cryptography, WCC 2011*, pages 193–202, Paris, France, 2011. 92

[Leg12] Matthieu Legeay. *Utilisation du groupe de permutations d'un code correcteur pour améliorer l'efficacité du décodage.* PhD thesis, Univ. Rennes 1, 2012. 92

[Leo82] J. Leon. Computing automorphism groups of error-correcting codes. *Information Theory, IEEE Transactions on*, 28(3):496–511, May 1982. 37

[Leo88] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988. 14, 29, 49

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite Fields.* Cambridge University Press, New York, NY, USA, 1997. 130

[LS01] P. Loidreau and N. Sendrier. Weak keys in the McEliece public-key cryptosystem. *IEEE Transactions on Information Theory*, 47(3):1207–1211, 2001. 6, 20, 53, 134

[LS12] Gregory Landais and Nicolas Sendrier. Implementing CFS. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2012. 8, 22, 42, 45

[Mac16] Francis S. Macaulay. *The Algebraic Theory of Modular Systems.* Cambridge University Press, 1916. 65

[MB09] Rafael Misoczki and Paulo S. L. M. Barreto. Compact McEliece Keys from Goppa Codes. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2009. iv, 6, 9, 10, 20, 21, 23, 24, 25, 81, 82, 83, 84, 88, 89, 91, 93, 105, 113, 115

[McE78] R. J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44. 4, 6, 18, 21, 47, 48

[Meu12] Alexander Meurer. *A Coding-Theoretic Approach to Cryptanalysis.* These, November 2012. 49

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding Random Linear Codes in $\tilde{O}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. 49

[MO15] Alexander May and Ilya Ozerov. On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes. In *Eurocrypt 2015*, Lecture Notes in Computer Science, page to appear, Sofia, Bulgaria, 2015. 49

[MRS00] C. Monico, J. Rosenthal, and A. Shokrollahi. Using low density parity check codes in the McEliece cryptosystem. In *Information Theory, 2000. Proceedings. IEEE International Symposium on*, pages 215–, 2000. 6, 20

[MS86] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North–Holland, Amsterdam, fifth edition, 1986. 4, 18, 33, 34, 35, 36, 37, 38, 39, 42, 84

[MS07] L. Minder and A. Shokrollahi. Cryptanalysis of the Sidelnikov cryptosystem. In *Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 347–360, Barcelona, Spain, 2007. 50

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2069–2073. IEEE, 2013. 6, 20

[NC11] Robert Niebuhr and Pierre-Louis Cayrel. Broadcast Attacks against Code-Based Schemes. In Frederik Armknecht and Stefan Lucks, editors, *WEWoRC*, volume 7242 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2011. 8, 23

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Probl. Control Inf. Theory*, 15:159–166, 1986. 5, 6, 19, 20, 50, 125

[NIKM08] Ryo Nojima, Hideki Imai, Kazukuni Kobara, and Kirill Morozov. Semantic security for the McEliece cryptosystem without random oracles. *Des. Codes Cryptography*, 49(1-3):289–305, 2008. 8, 23

[NMBB12] Robert Niebuhr, Mohammed Meziani, Stanislav Bulygin, and Johannes Buchmann. Selecting parameters for secure McEliece-based cryptosystems. *International Journal of Information Security*, 11(3):137–147, 2012. 83

[ÖOP03] Siddika Berna Örs, Elisabeth Oswald, and Bart Preneel. Power-Analysis Attacks on an FPGA - First Experimental Results. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2003. 7, 22

[OS09] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In DanielJ. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 95–145. Springer Berlin Heidelberg, 2009. 148

[Ove05] Raphael Overbeck. A New Structural Attack for GPT and Variants. In *Proceedings of the 1st International Conference on Progress in Cryptology in Malaysia*, Mycrypt'05, pages 50–63, Berlin, Heidelberg, 2005. Springer-Verlag. 50

[Ove08] R. Overbeck. Structural Attacks for Public Key Cryptosystems Based on Gabidulin Codes. *J. Cryptol.*, 21(2):280–301, February 2008. 6, 20, 50

[Pat75] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975. 39, 43

[PBH98] Vera Pless, Richard A. Brualdi, and W. C. Huffman. *Handbook of Coding Theory.* Elsevier Science Inc., New York, NY, USA, 1998. 4, 18, 49

[Per12a] Edoardo Persichetti. Compact McEliece keys based on quasi-dyadic Srivastava codes. *J. Mathematical Cryptology*, 6(2):149–169, 2012. 6, 20, 81, 82, 93, 116

[Per12b] Edoardo Persichetti. On a CCA2-secure variant of McEliece in the standard model. *IACR Cryptology ePrint Archive*, 2012:268, 2012. informal publication. 8, 23, 139

[Pet10] Christiane Peters. Information-set decoding for linear codes over Fq. In *PQCrypto 2010 [36] (2010), 81–94. URL: http://eprint.iacr.org/2009/589*, page 7, 2010. 49, 77

[RS60] I. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. 38

[Sen97] Nicolas Sendrier. On the Dimension of the Hull. *SIAM J. Discret. Math.*, 10(2):282–293, May 1997. 53

[Sen00] N. Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000. 6, 14, 20, 29, 37, 53, 54

[Sen02] Nicolas Sendrier. *Cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs.* PhD thesis, Université Paris 6, France, 2002. 7, 22, 54

[Sen10] Nicolas Sendrier, editor. *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, volume 6061 of *Lecture Notes in Computer Science.* Springer, 2010. 167

[Sha94] I. Shafarevich. *Basic Algebraic Geometry, Varieties in Projective Space*, volume Vol. 1. Springer Verlag, 2nd edition edition, 1994. 101

[Sho94a] 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*, pages 124–134. IEEE Computer Society, 1994. 5, 19

[Sho94b] Peter W. Shor. Polynominal time algorithms for discrete logarithms and factoring on a quantum computer. In Leonard M. Adleman and Ming-Deh A. Huang, editors, *Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings*, volume 877 of *Lecture Notes in Computer Science*, page 289. Springer, 1994. 5, 19

[Sid94] V.M. Sidelnikov. A public-key cryptosystem based on binary Reed-Muller codes. *Discrete Mathematics and Applications*, 4(3), 1994. 6, 20, 50

[SKHN75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding goppa codes. *Information and Control*, 27(1):87 – 99, 1975. 39, 44, 45

[SKHN76] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. Further results on Goppa codes and their applications to constructing efficient binary codes. *Information Theory, IEEE Transactions on*, 22(5):518–526, Sep 1976. 117

[SS92] V.M. Sidelnikov and S.O. Shestakov. On the insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 1(4):439–444, 1992. 6, 20, 50, 118, 125, 127

[SS99] N. Sendrier and G. Skersys. Permutation groups of error-correcting codes. In *WCC '99, Book of abstracts*, pages 33–42, Paris, France, January 1999. 37

[SS01a] D.V. Sarwate and N.R. Shanbhag. High-speed architectures for Reed-Solomon decoders. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(5):641–655, Oct 2001. 148

[SS01b] N. Sendrier and G. Skersys. On the Computation of the Automorphism Group of a Linear Code. In *IEEE Conference, ISIT 2001*, Washington D.C., USA, June 2001. 37

[SS13a] Nicolas Sendrier and Dimitrios E. Simos. How easy is code equivalence over Fq? In *WCC 2013 - International Workshop on Coding and Cryptography*, Bergen, Norvège, April 2013. 54

[SS13b] Nicolas Sendrier and Dimitris E. Simos. The Hardness of Code Equivalence over and Its Application to Code-Based Cryptography. In Gaborit [Gab13], pages 203–216. 54

[SSMS09] Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter, and Marc Stöttinger. A timing attack against patterson algorithm in the mceliece pkc. In Donghoon Lee and Seokhie Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 161–175. Springer, 2009. 8, 22, 137, 139, 140, 141, 142

[Ste88] J. Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988. 14, 29, 49

[Ste93] J. Stern. A New Identification Scheme Based on Syndrome Decoding. In D.R. Stinson, editor, *Advances in Cryptology - CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, 1993. 5, 19

[STM+08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side Channels in the McEliece PKC. In *Proceedings of the 2Nd International Workshop on Post-Quantum Cryptography*, PQCrypto '08, pages 216–229, Berlin, Heidelberg, 2008. Springer-Verlag. 139, 140

[Str10a] Falko Strenzke. A Smart Card Implementation of the McEliece PKC. In Pierangela Samarati, Michael Tunstall, Joachim Posegga, Konstantinos Markantonakis, and Damien Sauveron, editors, *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices*, volume 6033 of *Lecture Notes in Computer Science*, pages 47–59. Springer Berlin Heidelberg, 2010. 8, 22, 137

[Str10b] Falko Strenzke. A Timing Attack against the Secret Permutation in the McEliece PKC. In Sendrier [Sen10], pages 95–107. 8, 22, 137, 139, 143, 144, 145, 146

[Str13] Falko Strenzke. Timing Attacks against the Syndrome Inversion in Code-Based Cryptosystems. In Gaborit [Gab13], pages 217–230. 8, 13, 22, 28, 137, 143, 145, 146, 147, 148

[SWM+10] A. Shoufan, T. Wink, H.G. Molter, S.A. Huss, and E. Kohnert. A Novel Cryptoprocessor Architecture for the McEliece Public-Key Cryptosystem. *Computers, IEEE Transactions on*, 59(11):1533–1546, Nov 2010. 8, 22, 137

[SY09] D.V. Sarwate and Zhiyuan Yan. Modified Euclidean algorithms for decoding Reed-Solomon codes. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1398–1402, June 2009. 148

[Var97a] A. Vardy. The intractability of computing the minimum distance of a code. *Information Theory, IEEE Transactions on*, 43(6):1757–1766, Nov 1997. 35

[Var97b] Alexander Vardy. Algorithmic Complexity in Coding Theory and the Minimum Distance Problem. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 92–109, New York, NY, USA, 1997. ACM. 35

[Wie06a] C. Wieschebrink. Two np-complete problems in coding theory with an application in code based cryptography. In *Information Theory, 2006 IEEE International Symposium on*, pages 1733–1737, July 2006. 55

[Wie06b] Christian Wieschebrink. An Attack on a Modified Niederreiter Encryption Scheme. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 14–26. Springer Berlin Heidelberg, 2006. 6, 20, 50

[Wie10] Christian Wieschebrink. Cryptanalysis of the Niederreiter Public Key Scheme Based on GRS Subcodes. In Sendrier [Sen10], pages 61–72. 50, 51, 55

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. 63

[Yan11] Bo-Yin Yang, editor. *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*. Springer, 2011. 158, 162