UNIVERSITY OF TRENTO

DEPARTMENT OF MATHEMATICS

PHD IN MATHEMATICS

XXVIII CYCLE

Cyclic Codes: Low-Weight Codewords and Locators

Author:
Claudia TINNIRELLO

Advisor:
Prof. Massimiliano SALA

Head of the Doctoral School:
Prof. Valter MORETTI

March 2016

To Pietro, and to my family

Contents

Introduction

Error correcting codes has become an integral part of the design of reliable data transmissions and storage systems. Their employment in these applications provide mechanisms for the detection and correction of errors. Error correcting codes are also playing an increasingly important role for other applications such as the analysis of pseudorandom sequences and the design of secure cryptosystems. Cyclic codes form a large class of widely used error correcting codes, including important codes such as the Bose-Chaudhuri-Hocquenghem (BCH) codes, quadratic residue (QR) codes and Golay codes. Cyclic codes were studied first by E. Prange in 1957 [Pra57], who discovered their rich algebraic structure. Their extensive employment in real-life applications is due to two main aspects: they offer powerful error detection and correction capabilities and possess algebraic properties permitting the use of simplified processing procedures and a much easier investigation when compared to non-cyclic codes. For instance, the encoding of data into cyclic code words can easily be achieved in hardware using a simple linear feedback shift register. Currently, the main drawback of cyclic codes is the lack of an efficient decoding algorithm for them.

1.1 Objectives and Contributions

This thesis is devoted to two problems arising in the study of cyclic codes: finding low-weight codewords and decoding. In the rest of this section we briefly describe where our contribution is positioned and point out the main results of the thesis.

1.1.1 Finding Low-weight codewords

Computing efficiently low-weight codewords of a cyclic code is often a key ingredient of correlation attacks to LFSR-based stream ciphers. Correlation attacks were introduced by Siegenthaler in [Sie85] to cryptanalyze a large class of stream ciphers based on LFSRs. A major improvement by Meier and Staffelbach [MS89] led to different versions of fast correlation attacks [CT00, JJ99b, CJS01]. These attacks try to find a correlation between the output of the stream cipher and one of the LFSRs on which it is built, then they try to recover the state of the LFSR by decoding the keystream as a noisy version of the LFSR output. A fast version of a correlation attack involves the precomputation of multiple parity checks of one of the LFSRs in order to speed up the computation. This precomputation step can be performed according to two

different approaches, one based on the birthday paradox [CJM02] and another based on discrete logarithms [DLC07]. A cipher like E0 ([GPS04]) is not immediately subject to these types of attacks, since no apparently single LFSR is correlated to the keystream output. In [LV04], Lu and Vaudenay introduced a new fast correlation attack (often called faster correlation attacks) which is able to successfully recover the state of E0. Their attack requires a different precomputation step which computes a single parity check of multiple LFSRs. Since the data complexity of the attack is bounded from below by the degree of the parity check, one tries to find a parity check of degree less than a target degree. The complexity of their precomputation step is not far from the complexity of their full attack, and they employed the generalized birthday approach presented in [Wag02]. In Chapter 5 an alternative approach for solving this problem based on discrete logarithms is described. In some interesting cases our algorithm has a time complexity comparable to the generalized birthday approach, while having a much lower memory complexity (i.e. $O(1)$). These cases are relevant to faster correlation attacks to a class of stream ciphers (including the Bluetooth cipher E0) and when the polynomial is the product of many irreducible factors. The design of the new algorithm has been published [PST16].

1.1.2 Decoding cyclic codes

In the last fifty years many efficient bounded-distance decoders have been developed for special classes of cyclic codes, e.g. the Berlekamp-Massey (BM) algorithm ([Mas69]) designed for the BCH codes. Although BCH codes can be decoded efficiently, it is known that their decoding performance degrades as the length increases ([LW67]). Cyclic codes are not known to suffer from the same distance limitation, but no efficient bounded-distance decoding algorithm is known for them (up to their actual distance).
In [OS05] Orsini and Sala introduced the general error locator polynomial and presented an algebraic decoder which permits to determine the correctable error patterns of a cyclic code in one step. They constructively showed that these polynomials exist for any cyclic code (it is shown to exist in a Gröbner basis of the syndrome ideal), and gave some theoretical results on the structure of such polynomials in [OS07], without the need to actually compute a Gröbner basis. They also provide a structure theorem for these locators for a class of binary cyclic codes, which has been generalized by Chang and Lee [CL10] for binary cyclic codes that could be defined by only one syndrome. In Chapter 6 a generalization of this result is showed, along with several results for infinite classes of binary cyclic codes with $t = 2$ and $t = 3$. From these, a theoretically justification of the sparsity of the general error locator polynomial is obtained for all binary cyclic codes with $t \leq 3$ and $n < 63$, except for three cases where the sparsity is proved by a computer check. We study some consequences of these results to the understanding of the complexity of bounded-distance decoding of cyclic codes.

1.2 Thesis Organization

The remainder of the thesis is structures in three parts.

Part I: Background

In this part we aim to introduce and motivate our research objectives. We also provide some preliminary results which we will use in Part II. This part does not contain original contributions. It is organized as follows:

Chapter 2

In this chapter we review some well-known results in arithmetics, in complexity theory and in finite field theory that will be used along the thesis. Particular focus is directed towards fundamental properties of polynomials over finite fields and discrete logarithms computation.

Chapter 3

In this chapter the study of error correcting codes is introduced. Basic properties of linear codes are presented, and it is discussed the complexity of the decoding problem for these codes. The algebraic structure of cyclic codes is described and the problem of decoding cyclic codes is addressed. Established decoding techniques for BCH codes are presented along with their complexity. Finally, general error locators for cyclic codes are introduced, and it is discussed some promising properties of these locators for decoding cyclic codes.

Chapter 4

In this chapter basic notions in Cryptography and current algorithms for solving the Generalized Birthday Problem are briefly reviewed. Some basic properties of LFSR-based stream ciphers, their security level and the standard procedures for achieving a good level of security are described. Finally, correlation attacks and fast correlation attacks on LFSR-based stream ciphers are introduced.

Part II: Contributions

This part is devoted to our original results. It is organized as follows:

Chapter 5

In this chapter we describe an algorithm based on discrete logarithms for finding low-weight polynomial multiples of binary polynomials. First, the general strategy behind the algorithm is described and it is shown the algebraic results on which the algorithm is based. Then, the details of the algorithm are explained along with a comparison of its complexity to the generalized

birthday approach and to the straightforward generalization of the discrete log approach for the case of a single primitive polynomial. Significant examples of our approach are outlined in the final part of the chapter, where we show that for the fast correlation attack in [LV04] our algorithm could be more convenient to use in the second precomputation step than the generalized birthday approach, and that the method we propose is substantially better than the generalized birthday approach in the case where the polynomial can be decomposed in several irreducible factors, each one of degree less than 20. This chapter is based on a journal publication [PST16].

Chapter 6

In this chapter we deal with some issues concerning the efficiency of the Orsini-Sala bounded-distance decoding algorithm based on general error locators for cyclic codes. A new result on the structure of the general error locator polynomial for a class of cyclic codes is obtained, and it is shown sparse general error locator polynomials for infinite classes of binary cyclic codes with error correction capability $t \leq 3$, adding theoretical evidence to the sparsity of this locator for infinite classes of codes. Finally, the complexity of bounded-distance decoding of certain classes of cyclic codes is studied.

Part III: Appendices

This part contains two appendices.

- Appendix A includes two tables. The first table lists the binary cyclic codes with $t = 3$ and $n < 63$, while the second table reports a general error locator polynomial for binary cyclic codes with $t = 3$ and $n = 55$.
- Appendix B includes the Magma code implementing the algorithm in Chapter 5 and other procedures written in the Magma language that were used for obtaining the numerical results presented in Chapter 6 and in Appendix A.

Part I

Preliminary results

Algebraic and Complexity Background

In this chapter we fix some notations and report well-known results in arithmetics, in complexity theory and in finite field theory that will be used along the thesis.
For this chapter definitions and results are mainly from [LN97, Knu81, GJ79, MP13, MM07, MBG+13, Mor03].

2.1 Modular Arithmetic

In a Euclidean ring, we denote by $a \bmod c$ the remainder of the division of a by c. We write $a \equiv b \pmod{c}$ if a is congruent to b modulo c. We denote by LCM and GCD the Least Common Multiple and the Greatest Common Divisor — respectively — of polynomials or integers, depending on its inputs.

Definition 2.1.1. Let a and n be integers with $n > 0$ and $\mathrm{GCD}(a, n) = 1$. The smallest positive integer k with $a^k \equiv 1 \bmod n$ is called the *multiplicative order of* a modulo n and is denoted by $\mathrm{ord}_n(a)$.

Note that, as a consequence of Euler's theorem, $\mathrm{ord}_n(a)$ always divides $\varphi(n)$, φ being the Euler's function.

Definition 2.1.2. Let a and n be positive integers with $\mathrm{GCD}(a, n) = 1$, and let i be an integer with $0 \leq i < n$. The set

$$C_i = \{i, ia, ia^2, \ldots, ia^{s-1}\},$$

where s is the smallest positive integer with $ia^s \equiv i \pmod{n}$, is said the *cyclotomic coset of* a (or a*-cyclotomic coset*) *modulo* n *containing* i.

The distinct a-cyclotomic cosets modulo n partition the set of integers

$$[0, n-1] := \{0, 1, \ldots, n-1\}.$$

A subset $\{i_1, \ldots, i_s\}$ of $[0, n-1]$ is called a *complete set of representatives* of cyclotomic cosets of a modulo n if $C_{i_1}, \ldots, C_{i_s}$ are distinct and $\bigcup_{j=1}^{s} C_{i_j} = [0, n-1]$. Note that $\mathrm{ord}_n(a)$ is the size of the a-cyclotomic coset C_1 modulo n.

Next theorem is a basic result of Modular Arithmetics which provides an often-used technique to replace arithmetic on large integers with operations over small integers.

Theorem 2.1.3 (Chinese Remainder Theorem). *Let $m_1, m_2, \ldots, m_r$ be positive integers which are relatively prime in pairs, i.e.*

$$\mathrm{GCD}(m_j, m_k) = 1 \quad \textit{when } j \neq k.$$

Let $m = m_1 m_2 \cdots m_r$, and let $a, u_1, u_2, \ldots, u_r$ be any integers. Then, there is exactly one integer u which satisfies the conditions

$$a \leq u < a + m, \quad u \equiv u_j \pmod{m_j}, \quad 1 \leq j \leq r.$$

We are interested in the following generalization of Theorem 2.1.3.

Theorem 2.1.4 (Generalized Chinese Remainder Theorem). *Let $m_1, m_2, \ldots, m_r$ be positive integers. Let $m = \mathrm{LCM}(m_1, m_2, \ldots, m_r)$, and let $a, u_1, u_2, \ldots, u_r$ be any integers. Then, there is exactly one integer u which satisfies the conditions*

$$a \leq u < a + m, \quad u \equiv u_j \pmod{m_j}, \quad 1 \leq j \leq r$$

provided that

$$u_i \equiv u_j \pmod{\mathrm{GCD}(m_i, m_j)}, \quad 1 \leq i < j \leq r.$$

We will denote by $\mathrm{CRT}(u_1, u_2, \ldots, u_r, m_1, \ldots, m_r)$ the result of applying the (Generalized) Chinese Remainder Theorem to integers u_i and moduli m_i.

2.2 Some Notions in Complexity Theory

In this section we recall some notions from computational complexity theory. Since it is not within our scope to deal with details of computational complexity, most of our definitions will be informal. For more details we refer the reader to [GJ79].

2.2.1 Basic notions

The goal of computational complexity theory is to measure the difficulty of problems. By a *problem*, we mean a general question to be answered. A problem whose answer, or *solution*, is "yes" or "no", is called a *decision problem*. A problem usually possesses several *parameters* whose values are left unspecified. An *instance* of a problem is achieved by specifying values for those parameters.

Example 2.2.1. Consider the following problem

`Problem (PRIME)`
`Input`: An integer $n \geq 2$
`Question`: Is n prime?

The problem `PRIME` is an example of decision problem having one parameter n. An instance of this problem is obtained by specifying a value for n.

Notice that, given a problem, there are many ways in which its instances can be described. We assume that for each problem one particular way has been chosen in advance. The function which maps the problem instances into the strings describing them is called the *encoding scheme* of the problem. The *input length* for an instance I of a problem Π is defined to be the number of symbols in the description of I obtained from the encoding scheme for Π.
A general step-by-step procedure, solving a problem, in a finite number of steps is called an *algorithm*.
The difficulty of a problem is measured by examining algorithms to solve it. The (worst-case) *time complexity* for an algorithm expresses its time requirements by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size. The time complexity function of an algorithm is usually denoted by $T(n)$, with n the input length.
Note that, depending on the assumptions, may be necessary consider separately the *space complexity* of an algorithm, which is the number of registers used in the course of the algorithm. This is not essential for single-tape Turing machine, whose running time combines the time and memory complexity. A very significant fact about Turing machines is that any algorithm can be modeled by a Turing machine program. However, for a general algorithm writing such a program is a notably time-consuming task.

Different algorithms may have very different time complexity functions. One would like to characterize which of these algorithms are "too inefficient", or " intractable", and which are "tractable". Computer scientists recognize a simple distinction that offers considerable insight into these issues.

Definition 2.2.2 (Big O Notation)**.** Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be two positive real-valued functions. We say that f is $O(g(n))$ if there a positive constant c and an integer n_0 such that

$$f(n) < c \cdot g(n), \tag{2.1}$$

for all values of $n \geq n_0$.

A *polynomial time algorithm* is defined to be one whose time complexity function is $O(p(n))$ for some polynomial function p, where n denotes the input length. Any algorithm whose time complexity function cannot be so bounded is called an *exponential time algorithm*. We shall refer to a problem as *intractable* if no polynomial time algorithm can possibly solve it.

A *randomized algorithm* is one wherein certain decisions are made based on the outcomes of coin flips made in the algorithm. By *probabilistic polynomial time algorithm*, we mean a randomized algorithm whose time complexity function is bounded by a polynomial in the size of the input.

Definition 2.2.3. We define

$$L[n, \gamma, c] := O(\exp((c + o(1))(\ln n)^{\gamma}(\ln \ln n)^{1-\gamma}))),$$

where n is the size of the input space, $0 \leq \gamma \leq 1$, c is a constant and $\ln n$ denotes the natural logarithm.

Let $\mathcal{A}$ be an algorithm. Denote with $T(n)$ its time complexity function. We note that if $T(n) = L[n, 0, c]$ then $\mathcal{A}$ is a polynomial time algorithm, while if $T(n) = L[n, 1, c]$ then $\mathcal{A}$ is an exponential time algorithm. We say that $\mathcal{A}$ is a *subexponential time algorithm* if $T(n) = L[n, \gamma, c]$ with $0 < \gamma < 1$.

2.2.2 Theory of NP-Completeness

The theory of complexity is designed to be applied only to decision problems. The reason is that they possess useful properties allowing a better understanding of their complexity than non-decision problems. Moreover, most problems can be reduced to decision problems.
An important class of decision problems is the class P. We say that a decision problem Π belongs to P if Π is a polynomial-time algorithm. Another important class of decision problems is the class NP. In order to introduce this class let us first consider the following example.

Example 2.2.4. The problem PRIME in Example 2.2.1 is an example of problem in the class P. A (deterministic) polynomial time algorithm solving it, is the AKS primality test [AKS04]. Let us examine the following problem

`Problem (THREE DIMENSIONAL MATCHING)`
`Input`: A subset $U \subseteq T \times T \times T$, where T is a finite set
`Question`: Is there a set $W \subseteq U$ such that $|W| = |T|$, and no two elements of W agree in any coordinate?

No polynomial time algorithm is known for solving the `THREE DIMENSIONAL MATCHING`. However, suppose someone claimed, for a particular instance (T, U) of this problem, that the answer for that instance is "yes" providing us with a set W which satisfies the property. It is reasonable to expect that the problem of verifying the truth or falsity of such a claim, would be easier than the problem itself. It can be seen that for the `THREE DIMENSIONAL MATCHING` this verification problem can be solved with a polynomial time algorithm. The class NP captures this notion of polynomial time verifiability.

An algorithm is said *deterministic* if its computation is fully determined by its input. An algorithm is *nondeterministic* if it is not deterministic, that is, if there may be more than one computation for a given input.
A nondeterministic algorithm is composed of two stages:

1. *Guessing stage* (nondeterministic): Given a problem instance I, some structure S is "guessed". S can be thought of as a candidate solution of the problem for the instance I.

2. *Verification stage* (deterministic): Taken as input I and S, it returns "yes" if the candidate solution represents actual solution for the instance I.

Let Π be a decision problem. Denote with D_Π the set of instances of Π. We say that an instance $I \in D_\Pi$ is an *yes-instance* of Π if the solution of Π when particularized to the instance I is "yes". We denote the set of all yes-instances of Π with Y_Π.
A nondeterministic algorithm is said to solve a decision problem Π if for all instances $I \in D_\Pi$:

- If $I \in Y_\Pi$, then there exists some structure S that, when guessed for input I, will lead the verification stage to respond "yes" for I and S.
- If $I \notin Y_\Pi$, then there exists no structure S that, when guessed for input I, will lead the verification stage to respond "yes" for I and S.

In the complexity investigation of a decision problem Π, one is usually interested in showing that a polynomial-time algorithm exists for Π or that Π is NP-complete.
A decision problem Π belonging in NP is said an *NP-complete problem* if a polynomial-time algorithm for Π would yield a polynomial-time algorithm for every decision problem in the class NP. A decision problem which has this property but is not necessary in NP is called an *NP-hard problem*.
The concept of NP-completeness was introduced in 1971 by Cook [Coo71] which also provided the first known NP-complete problem. Since the class NP is known to contain many problems that are considered computationally hard, the NP-completeness of a problem Π is considered as a strong evidence suggesting that Π does not admit a polynomial-time algorithm. Today a wide variety of problems are known to be NP-complete. Note that, once we have at least one known NP-complete problem available, a straightforward method for proving that a decision problem Π is NP-complete consists in showing that

- $\Pi \in NP$;
- some known NP-complete problem Π' can be transformed with a polynomial time transformation in Π.

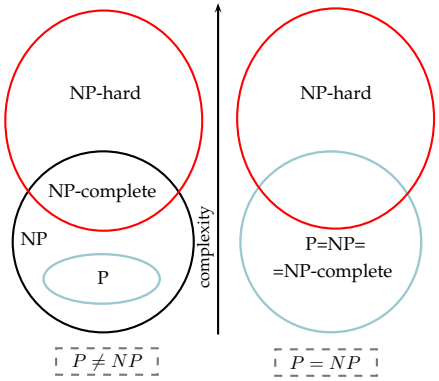

Figure 2.1: Euler diagram for complexity classes P, NP, NP-complete and NP-hard under the assumption that $P \neq NP$ (on the left) and $P = NP$ (on the right).

Note also that, trivially, $P \subseteq NP$. On the contrary, whether or not $NP \subseteq P$ is one of the major unsolved problems in computer science. Figure 2.1 shows a diagram of complexity classes under the two assumptions that $P \neq NP$ and $P = NP$.

2.3 Basic Notions in Finite Fields

We denote by $\mathbb{F}_q$ the field of q elements, where q is a power of a prime p, by $\mathbb{F}_q^*$ the multiplicative group of non-zero elements of $\mathbb{F}_q$, by $\overline{\mathbb{F}}_q$ the algebraic closure of $\mathbb{F}_q$, and by $\mathbb{F}_q^n$ the standard n-dimensional vector space over $\mathbb{F}_q$.
In this section we recall some of the basic properties of finite fields.

Definition 2.3.1. A subset $\mathbb{K}$ of a field $\mathbb{F}$ that is a field under the operations of $\mathbb{F}$ is called a *subfield* of $\mathbb{F}$, and we write $\mathbb{K} \subseteq \mathbb{F}$. In this context, $\mathbb{F}$ is a called an *extension* of $\mathbb{K}$. If M is any subset of $\mathbb{F}$, then the field $\mathbb{K}[M]$ is defined as the intersection of all subfields of $\mathbb{F}$ containing both $\mathbb{K}$ and M. For finite $M = \{\theta_1, \ldots, \theta_n\}$, we write $\mathbb{K}(\theta_1, \ldots, \theta_n)$.

Proposition 2.3.2. *Let $\mathbb{F}_q$ be a finite field with $q = p^s$. Every subfield of $\mathbb{F}_q$ has p^m elements for some integer m dividing s. Conversely, for any integer m dividing s there exists a unique subfield of $\mathbb{F}_q$ with p^m elements.*

Let $\mathbb{F}_q[x]$ be the ring of univariate polynomials over $\mathbb{F}_q$.

Definition 2.3.3. Let $f \in \mathbb{F}_q[x]$. The coefficient of the highest power of x in f is called the *leading coefficient* of f. We say that f is *monic* if its leading coefficient is 1.

Definition 2.3.4. Let $\theta \in \mathbb{F}_{q^r}$ and $f(\theta) = 0$ where $f(x)$ is a monic polynomial in $\mathbb{F}_q[x]$. Then $f(x)$ is called the *minimal polynomial* of θ over $\mathbb{F}_q$ if θ is not a root of any nonzero polynomial in $\mathbb{F}_q[x]$ of lower degree.

Definition 2.3.5. A polynomial $f \in \mathbb{F}_q[x]$ is an *irreducible polynomial* over $\mathbb{F}_q$ if f has a positive degree and $f = gh$ with $g, h \in \mathbb{F}_q$ implies that either g or h is a constant polynomial.

Proposition 2.3.6. *Let $\theta \in \mathbb{F}_{q^r}$ and $f(x)$ be its minimal polynomial over $\mathbb{F}_q$. Then f is an irreducible polynomial.*

Definition 2.3.7. Let f be a non-zero polynomial in $\mathbb{F}_q[x]$. If $f(0) \neq 0$, then the order of f is the least positive integer N such that $1 + x^N$ is a multiple of f, and we denote it by $\mathrm{ord}(f)$.

Proposition 2.3.8. *Let f be a polynomial in $\mathbb{F}_q[x]$ of positive degree and $f(0) \neq 0$. Let $f = f_1^{b_1} \cdots f_r^{b_r}$, where $b_1, \ldots, b_r \in \mathbb{N}$ and $f_1, \ldots, f_r$ are distinct irreducible polynomials of degree $n_1, \ldots, n_r$, be the factorization of f in $\mathbb{F}_q[x]$. Then,*

1. $\mathrm{ord}(f) = q^t \,\mathrm{LCM}(\mathrm{ord}(f_1), \ldots, \mathrm{ord}(f_r))$ *where t is the smallest integer such that q^t is bigger or equal than $\max(b_1, \ldots, b_r)$;*

2. *If f_i is irreducible, then* $\mathrm{ord}(f_i) \mid q^{n_i} - 1$*;*

Definition 2.3.9. Let $f \in \mathbb{F}_q[x]$ of degree n. We say that f is *primitive* if $\mathrm{ord}(f) = q^n - 1$.

Note that by point 2. of Proposition 2.3.8 we have that any primitive polynomial is irreducible.

Definition 2.3.10. Let $f \in \mathbb{F}_q[x]$ be monic and of positive degree and $\mathbb{F}$ be an extension of $\mathbb{F}_q$. Then f is said to *split in* $\mathbb{F}$ if can be factored as a product of linear factors in $\mathbb{F}[x]$, i.e. $f(x) = (x - \alpha_1) \cdots (x - \alpha_n)$. The field $\mathbb{F}$ is a *splitting field* of f over $\mathbb{F}_q$ if f splits in $\mathbb{F}$ and $\mathbb{F} = \mathbb{F}_q(\alpha_1, \ldots, \alpha_n)$.

Proposition 2.3.11 (Existence and Uniqueness of Splitting Field)**.** *Let $f \in \mathbb{F}_q[x]$ be monic and of positive degree. Then there exists a splitting field of f over $\mathbb{F}_q$. Moreover, any two splitting fields of f over $\mathbb{F}_q$ are isomorphic.*

Next theorem summarizes some fundamental properties of finite fields

Theorem 2.3.12. *Let $\mathbb{F}_q$ be a finite field with $q = p^s$. Then*

1. $(a + b)^{p^k} = a^{p^k} + b^{p^k}$ *for* $a, b \in \mathbb{F}_q$ *and* $k \in \mathbb{N}$
2. *every* $a \in \mathbb{F}_q$ *satisfies* $a^q = a$*.*
3. $x^q - 1$ *factors in* $\mathbb{F}_q$ *as* $\prod_{a \in \mathbb{F}_q^*}(x - a)$
4. $\mathbb{F}_q$ *is isomorphic to the splitting field of* $x^q - x$ *over* $\mathbb{F}_p$
5. $\mathbb{F}_q^*$ *is a cyclic group*

Definition 2.3.13. An element $\theta \in \mathbb{F}_q$ which multiplicatively generates the group $\mathbb{F}_q^*$ is called a *primitive element*.

Proposition 2.3.14. *Let $\mathbb{F}_q$ be a finite field, n be an integer with* $\mathrm{GCD}(n, q) = 1$*, and $\mathbb{F}_{q^m}$ be the splitting field of $x^n - 1$ over $\mathbb{F}_q$. Then there exists an element $\alpha \in \mathbb{F}_{q^m}$ such that*

$$x^n - 1 = \prod_{i=0}^{n-1}(x - \alpha^i). \tag{2.2}$$

A such element α is called a primitive nth root on unity over $\mathbb{F}_q$.

Definition 2.3.15. Let $\mathbb{F}_{q^m}$ be an extension of $\mathbb{F}_q$. An automorphism of $\mathbb{F}_{q^m}$ is said to be an *automorphism of* $\mathbb{F}_{q^m}$ *over* $\mathbb{F}_q$ if it fixes the elements of $\mathbb{F}_q$. If $\alpha \in \mathbb{F}_{q^m}$, then the elements $\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}$ are called the *conjugates* of α with respect to $\mathbb{F}_q$.

Conjugate elements in $\mathbb{F}_{q^m}$ with respect to $\mathbb{F}_q$ are related to the automorphisms of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ as the following proposition shows.

Proposition 2.3.16. *The distinct automorphisms of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ are exactly the functions $f_0, f_1, \ldots, f_{m-1}$, where*

$$f_j(\alpha) = \alpha^{q^j} \quad \text{for } \alpha \in \mathbb{F}_{q^m},$$

for $0 \leq j \leq m-1$. The function f_1 is called the Frobenius automorphism of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$.

Theorem 2.3.17. *Let $\mathbb{F}_q$ be a finite field and $\mathbb{F}_{q^m}$ be the splitting field of $x^n - 1$ over $\mathbb{F}_q$. Let $\alpha \in \mathbb{F}_{q^m}$ be a primitive nth root of unity over $\mathbb{F}_q$ with $\mathrm{GCD}(n, q) = 1$. Then*

- *For each integer i with $0 \leq i < n$, the minimal polynomial of a^i over $\mathbb{F}_q$ is*

$$m_{\alpha^i}(x) = \prod_{j \in C_i} (x - \alpha^j) \in \mathbb{F}_q[x],$$

  where C_i is the q-cyclotomic coset modulo n containing i.

- *The conjugates of α^i are the elements α^s with $s \in C_i$.*

- *The polynomial $x^n - 1$ has the factorization into monic irreducible polynomials over $\mathbb{F}_q$*

$$x^n - 1 = \prod_i m_{\alpha^i}(x),$$

  where i runs through a set of representatives of the q-cyclotomic cosets modulo n.

An important property of finite fields is that every function defined on a finite field can be realized by a polynomial with coefficients in that field. It is remarkable that this property characterizes finite fields in the sense finite fields are the only commutative ring with identity satisfying it. The next result tell us how to obtain a polynomial representing a given function over a field.

Theorem 2.3.18 (Lagrange Interpolation Formula)**.** *For $n \geq 0$, let $a_0, a_1, \ldots, a_n$ be $n+1$ distinct elements of a field $\mathbb{F}$, and let $b_0, b_1, \ldots, b_n$ be $n+1$ arbitrary elements in $\mathbb{F}$. Then there exists exactly one polynomial $f \in \mathbb{F}[x]$ of degree $\leq n$ such that $f(a_i) = b_i$ for $i = 0, \ldots, n$. This polynomial is given by*

$$f(x) = \sum_{i=0}^{n} b_i \prod_{\substack{k=0 \\ k \neq j}}^{n} (a_i - a_k)^{-1}(x - a_k).$$

For finite fields we can do somewhat better.

Theorem 2.3.19. *Every function $f : \mathbb{F}_q \to \mathbb{F}_q$ can be represented by a unique polynomial over $\mathbb{F}_q$ of degree at most $q-1$, i.e there exist exactly one polynomial $P \in \mathbb{F}_q[x]$ such that $P(a) = f(a)$ for all $a \in \mathbb{F}_q$. Moreover such a polynomial is given by*

$$\sum_{a \in \mathbb{F}_q} f(a) \left(1 - (x - a)^{q-1}\right).$$

The previous result can be extended to functions with any number of variables.

Theorem 2.3.20. *For any integer $r \geq 1$, let $f : \mathbb{F}_q^r \to \mathbb{F}_q$. Then f can be represented by a unique polynomial $P \in \mathbb{F}_q[x_1, \ldots, x_r]$ of degree at most $q-1$ in each variable. Moreover, such a polynomial is given by*

$$\sum_{(a_1,\ldots,a_r)\in\mathbb{F}_q^r} f(a_1,\ldots,a_r) \prod_{1\leq i\leq r} \left(1-(x_i-a_i)^{q-1}\right).$$

Definition 2.3.21. Let $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ given by $f(x_1, \ldots, x_n) = \sum a_{i_1\ldots i_n} x_1^{i_1} \cdots x_n^{i_n}$ with $n \geq 1$. If $a_{i_1\ldots i_n} \neq 0$, then $a_{i_1\ldots i_n} x_1^{i_1} \cdots x_n^{i_n}$ is called a *term* of f and $i_1 + \cdots + i_n$ is called the degree of the term. For $f \neq 0$, we define the *degree* of f, denoted by $\deg(f)$, as the maximum of the degrees of the terms of f. For $f = 0$ we set $\deg(f) = -\infty$. If $f = 0$ or if all terms of f have the same degree, then f is called *homogeneous*.

An important class of polynomials in n variables we will treat in Chapter 6 is that of symmetric polynomials.

Definition 2.3.22. Let $f \in \mathbb{F}_q[x_1, \ldots, x_n]$. We say that f is *symmetric* if $f(x_{i_1}, \ldots, x_{i_n}) = f(x_1, \ldots, x_n)$ for any permutation $i_1, \ldots, i_n$ of the integers $1, \ldots, n$.

Example 2.3.23. Let z be a variable over $\mathbb{F}_q[x_1, x_2, \ldots, x_n]$, and let $g(z) = (z - x_1)(z - x_2) \cdots (z - x_n)$. Then

$$g(z) = z^n - \sigma_1 z^{n-1} + \sigma_2 z^{n-2} + \cdots + (-1)^n \sigma_n, \tag{2.3}$$

where

$$\sigma_k = \sigma_k(x_1, \ldots, x_n) = \sum_{1\leq i_1<\cdots<i_k\leq n} x_{i_1} \cdots x_{i_k}, \quad k = 1, 2, \ldots, n. \tag{2.4}$$

Since g remains unchanged under any permutation of the x_j, then the polynomials σ_k are symmetric polynomials. Moreover, by definition, they are homogeneous.

Definition 2.3.24. For $k = 1, 2, \ldots, n$, the polynomial σ_k defined by (2.4) is called the kth *elementary symmetric polynomial* in the variables $x_1, \ldots, x_n$ over $\mathbb{F}_q$.

Theorem 2.3.25 (Fundamental Theorem on Symmetric Polynomials)**.** *For any symmetric polynomial $f \in \mathbb{F}_q[x_1 \ldots, x_n]$ there exists a uniquely determined polynomial $h \in \mathbb{F}_q[x_1, \ldots, x_n]$ such that*

$$f(x_1, \ldots, x_n) = h(\sigma_1, \ldots, \sigma_n).$$

Definition 2.3.26. For $k \geq 1$, the polynomial $x_1^k + \ldots + x_n^k \in \mathbb{F}_q[x_1, \ldots, x_n]$ is called the *kth power sum polynomial* in the variables $x_1, \ldots, x_n$ over $\mathbb{F}_q$.

Theorem 2.3.27 (Newton's identities)**.** *Let $\sigma_1, \ldots, \sigma_n$ be the elementary symmetric polynomials in the variables $x_1, \ldots, x_n$ over $\mathbb{F}_q$, and let $s_0 = n \in \mathbb{Z}$ and $s_k = s_k(x_1, \ldots, x_n)$ be the kth power sum polynomials in the variables $x_1, \ldots, x_n$ over $\mathbb{F}_q$ for $k \geq 1$. Then the following formula holds for $k \geq 1$*

$$s_k - s_{k-1}\sigma_1 + s_{k-2}\sigma_2 + \cdots + (-1)^{m-1}s_{k-m+1}\sigma_{m-1} + (-1)^m \frac{m}{n} s_{k-m}\sigma_m = 0,$$

where $m = \min(k, n)$.

Theorem 2.3.28 (Waring's Formula)**.** *With the same notation as Theorem 2.3.27, for $k \geq 1$, we have that*

$$s_k = \sum (-1)^{i_2+i_4+i_6+\cdots} \frac{(i_1 + i_2 + \cdots i_n - 1)!}{i_1! i_2! \cdots i_n!} k \sigma_1^{i_1} \sigma_2^{i_2} \cdots \sigma_n^{i_n},$$

where the summation is extended over all n-tuples $(i_1, \ldots, i_n)$ of non-negative integers with $i_1 + 2i_2 + \cdots + ni_n = k$. The coefficient of $\sigma_1^{i_1}\sigma_2^{i_2} \cdots \sigma_n^{i_n}$ is always an integer.

We conclude this section by recalling a classical technique, due to Kronecker [Kro82], for reducing problems concerning multivariate polynomials to the case of univariate polynomials. While this technique can be applied to any ring we describe it only for finite fields.

Let $\mathbb{F}_q$ be a finite field, and let $\mathcal{P} = \mathbb{F}_q[x_1, \ldots, x_n]$. For $f \in \mathcal{P}$, let us denote by $\deg_i f$ the degree of f in the variable x_i. Moreover, for $d \in \mathbb{N}$, let

$$\mathcal{P}_d := \{f \in \mathbb{F}_q[x_1, \ldots, x_n] | \deg_i f < d, \text{ for all } i\}.$$

Definition 2.3.29. The map $\chi_d : \mathcal{P} \to \mathbb{F}_q[x]$ defined by

$$\chi_d(F(x_1, x_2, \ldots, x_n)) = F(x, x^d, \ldots, x^{d^{n-1}})$$

is called *Kronecker's substitution*.

Theorem 2.3.30. *With the above notation, we have that the restriction $\overline{\chi}_d$ of the map χ_d to $\mathcal{P}_d$ is a $\mathbb{F}_q$-vector space isomorphism between $\mathcal{P}_d$ and its image in $\mathbb{F}_q[x]$, which is*

$$\mathrm{Im}(\overline{\chi}_d) = \{f \in \mathbb{F}_q[x] | \deg f \leq d^n - 1\}.$$

The previous result is a straightforward consequence of the fact that every non-negative integer a is uniquely represented as $\sum_{i=1}^{n} a_i d^{i-1}$, where $0 \leq a_i < d$. If $f = \sum_j c_j x^{a^{(j)}}$, then

$$\overline{\chi}_d^{-1}(f) = \sum_j c_j x_1^{a_1^{(j)}} \cdots x_n^{a_n^{(j)}},$$

where $a_1^{(j)} + a_2^{(j)} d + \cdots a_n^{(j)} d^{n-1}$ is the d-adic representation of $a^{(j)}$.

Kronecker substitution is rarely mentioned in modern algebra textbooks. However, it is an extensively used technique in practice. For instance the Magma computer algebra system [BCP97] uses Kronecker substitution for multiplying polynomials in $\mathbb{Z}[x]$ with high degree and small coefficients. An improvement of the original Kronecker's method for multiplying in $\mathbb{Z}[x]$ is due to Harvey [Har09]. Kronecker's substitution can be also used as a basis of an algorithm for factoring polynomials [Mor03, p. 436].

2.4 Linear Recurring Sequences

Sequences of finite fields elements whose terms depend in a simple manner on their predecessors are employed in several applications. In many of these applications the underlying field is often $\mathbb{F}_2$, but the results can be extended quite generally for any finite field. Of particular interest for the applications is the case where the terms depend linearly on a fixed number of predecessors. These sequences are called *linear recurring sequences*.

Definition 2.4.1. Let k be a positive integer, and $a_0, a_1, \ldots, a_{k-1}$ be given elements of a finite field $\mathbb{F}_q$. A sequence $s_0, s_1, \ldots$ of elements of $\mathbb{F}_q$ satisfying the relation

$$s_{i+k} = a_{k-1}s_{i+k-1} + a_{k-2}s_{i+k-2} + \cdots + a_1 s_{i+1} + a_0 s_i + a \quad \text{for } i = 0, 1, \ldots \tag{2.5}$$

is called a (k-order) linear recurring sequence in $\mathbb{F}_q$. We call the terms $s_0, s_1, \ldots, s_{k-1}$ that uniquely determine the rest of the sequence, the *initial values* of the sequence, or the *initial state* of the sequence if we refers to the vector $(s_0, s_1, \ldots, s_{k-1})$. If $a = 0$ then the sequence is called *homogeneous*, otherwise it is called *inhomogeneous*.

From now on, if not differently specified, we consider only homogeneous linear recurring sequences.

A common way to implement the generation of linear recurring sequences is to use *linear feedback shift registers* (LFSR). LFSRs are useful tools both in coding theory and in cryptography. In particular they are one of the most useful devices in the generation of keystreams.
A linear feedback shift register consists of two parts: a shift register of length k and a feedback function F. The shift register is a register with k cells each containing an element of $\mathbb{F}_q$. The content of the k cells forms the *state* of the LFSR. The feedback function is usually the XOR of the content of certain cells, which are called *taps*. The first output sequence element is the least significant element of the initial state, i.e the element in the rightmost cell of the shift register. Then, the shift register is shifted by one cell to the right and the new leftmost cell is filled with the XOR of the taps. By iterating the previous procedure, the LFSR produces a semi-infinite sequence of elements in $\mathbb{F}_q$. Figure 2.2 shows a general LFSR in the Fibonacci representation.

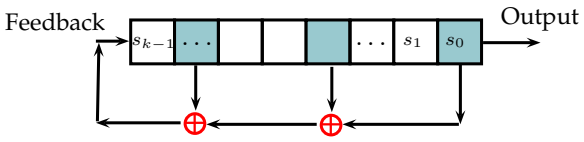

Figure 2.2: Linear feedback shift register

In the following we recall some basic results concerning linear recurring sequences.

Definition 2.4.2. Let S be an arbitrary nonempty set, and let $s_0, s_1, \ldots$ be a sequence of elements of S. The sequence $s_0, s_1, \ldots$ is *periodic* if there exists an integer $r > 0$ such that $s_{n+r} = s_n$ for all $n = 0, 1, \ldots$.

Proposition 2.4.3. *If $s_0, s_1, \ldots$ is a kth-order linear recurring sequence in a finite field $\mathbb{F}_q$ satisfying the linear recurrence relation* (2.5)*, and if the coefficient a_0 of* (2.5) *is nonzero, then the sequence $s_0, s_1, \ldots$ is periodic and its period r is $\leq q^k$.*

Let $s_0, s_1, \ldots$ be an homogeneous linear recurring sequence in $\mathbb{F}_q$ satisfying

$$s_{i+k} = a_{k-1}s_{i+k-1} + a_{k-2}s_{i+k-2} + \cdots + a_1 s_{i+1} + a_0 s_i \quad \text{for } i = 0, 1, \ldots, \tag{2.6}$$

where $a_j \in \mathbb{F}_q$, for $0 \leq j \leq k-1$. To this linear recurring sequence we associate the following $k \times k$ matrix over $\mathbb{F}_q$

$$A = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 & a_0 \\ 1 & 0 & 0 & \ldots & 0 & a_1 \\ 0 & 1 & 0 & \ldots & 0 & a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & a_{k-1} \end{pmatrix}. \tag{2.7}$$

We note that the matrix A depends only on the linear recurrence relation satisfied by the given sequence. In particular it does not depend on the initial state of the sequence. The polynomial

$$f(x) = x^k - a_{k-1}x^{k-1} - a_{k-2}x^{k-2} - \cdots - a_0 \in \mathbb{F}_q[x]$$

is said the *characteristic polynomial* of the linear recurring sequence, and, as the matrix A, it depends only on the linear recurrence relation (2.6). It is easy to see that $f(x)$ corresponds to the characteristic polynomial of A in the sense of linear algebra. Linear recurring sequences whose periods are very large are of particular interest in applications. Next Theorem suggests how to produce such sequences.

Theorem 2.4.4. *Let $s_0, s_1, \ldots$ be an homogeneous linear recurring sequence in $\mathbb{F}_q$ with nonzero initial state, and suppose its characteristic polynomial $f(x) \in \mathbb{F}_q$ is irreducible over $\mathbb{F}_q$ and satisfies $f(0) \neq 0$. Then the sequence is periodic with period equal to* $\mathrm{ord}(f(x))$.

By Theorem 2.4.4 it follows that kth-order maximal period sequences in $\mathbb{F}_q$ are those whose characteristic polynomial is a primitive polynomial over $\mathbb{F}_q$, and its period is equal to the largest possible value for the period of any kth-order linear recurring sequences in $\mathbb{F}_q$ – namely $q^k - 1$. Sequences with maximal period have also many desirable statistical properties. The best known of these properties is to satisfy Golomb's axioms for pseudo-random sequences [G+82].

2.5 Discrete Logarithms

Let $K = \mathbb{F}_q$ and $F = \mathbb{F}_{q^m}$. Usually F is represented as an m-dimensional vector space over K, so additions in F become trivial given the arithmetics in K. However, the choice of a basis of F over K is crucial for efficiently performing multiplication, inversion and exponentiation. Various types of bases have been studied extensively. Among these there are two special types

of bases of particular importance. The first is a *polynomial basis* $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ where α is a root of an irreducible polynomial of degree m over K. In this context, one often prefers α to be a primitive element of F. The second basis is a *normal basis*, that is, a basis of the form $\{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\}$.

An alternative to using basis representations is to represent the non-zero elements of F as the powers of a primitive element $\alpha \in F$. In this case multiplication in F is trivial, but addition then becomes difficult. In practical applications where repeated computations over a relatively small finite field are required, this problem can be overcome using discrete logarithms in conjunction with Zech logarithms. In addition to this employment, discrete logarithms in finite fields play an important rule in cryptanalysis.

In this subsection we summarize the main current algorithms for computing discrete logarithms in finite fields and their use to improve arithmetic in small finite fields.

Definition 2.5.1 (Discrete Logarithm)**.** Let $\alpha \in \mathbb{F}_q$ a primitive element and $\beta \in \mathbb{F}_q, \beta \neq 0$. We define the discrete logarithm of β with respect to α as the unique integer $0 \leq i < q$ such that $\alpha^i = \beta$. We use the notation $i = \mathrm{DLog}_\alpha(\beta)$.

2.5.1 Computing Discrete Logarithms

The most obvious method for finding discrete logarithms in $\mathbb{F}_q$ is to precompute a table of logarithms once and for all time. Another one is to successively compute consecutive powers of α and compare with β until a match is found. Both methods become computationally infeasible when q is sufficiently large, since they cost $O(q)$. A more interesting method, but still not very practical, to compute discrete logarithms in $\mathbb{F}_q$ with $q = p^s$ is given by Mullen and White in [MW86] and it consists in exhibiting a polynomial representation for the DLog function modulo p. They prove the following result

Proposition 2.5.2. *Let α be a generator for $\mathbb{F}_q$ with $q = p^s$. For any $\beta = \alpha^i \in \mathbb{F}_q^*$, $q \geq 3$, we have that*

$$i \equiv -1 + \sum_{k=1}^{q-2} \frac{\beta^k}{\alpha^{-k} - 1} \pmod{p}$$

The two main kind of methods used to compute discrete logarithms in finite fields are the *Pohlig-Hellman method* combined with the *Baby-step Giant-step algorithm* or with the *Pollard-ρ algorithm*, and the *Index-Calculus method*. In practice, many computational algebra systems use the Pohlig-Hellman method for computing discrete logarithms in $\mathbb{F}_q$ when the maximal prime factor of q is approximately less than 2^{36}, and they use Index-Calculus method in the other cases.
In the following we briefly describe these algorithms.

Baby-step Giant-step algorithm

The first algorithm we describe is the Baby-step Giant-step algorithm attributed to Shanks. The algorithm works as follows. Let r, s be integers with $r \cdot s > q$. Precompute a list of pairs, (i, α^i) for $0 \leq i < r$ and sort this list by second component. These precomputations are called *baby-steps*. The giant steps consist on computing for each j, $0 \leq j < s$, the element $\beta\alpha^{-jr}$ and see (by a binary search) if this element is the second component of some pair in the list. If $\beta\alpha^{-jr} = \alpha^i$ for some i, $0 \leq i < r$, then $\mathrm{DLog}_\alpha(\beta) = q$ if $i = j = 0$ and $\mathrm{DLog}_\alpha(\beta) = i + jr$ otherwise. The baby steps require a table with $O(r)$ entries, while to sort the table and search the table for each value of j requires $O(r+s)$ field operations. Usually one choose $r = s = \sqrt{q-1}$ in order to get $O(\sqrt{q-1})$ for both time and memory complexity.

Pollard-ρ method

One drawback of Shanks algorithm is the need to sort and store a list of size $\sqrt{q-1}$. Pollard [Pol78] introduced a probabilistic algorithm which eliminates the need for such storage. Partition $\mathbb{F}_q^*$ into three sets S_1, S_2, S_3 of roughly equal size. Consider the sequence of elements in $\mathbb{F}_q^*$ $x_0, x_1, x_2, \ldots$ with $x_0 = 1$ and

$$x_i = \begin{cases} \beta x_{i-1} & x_{i-1} \in S_1 \\ x_{i-1}^2 & x_{i-1} \in S_2 \\ \alpha x_{i-1} & x_{i-1} \in S_3 \end{cases}$$

It is easy to see that this sequence defines two sequences of integers $\{a_i\}$ and $\{b_i\}$ where $x_i = \beta^{a_i}\alpha^{b_i}$, $i \geq 0$, $a_0 = b_0 = 0$, $a_{i+1} \equiv a_i + 1$, $2a_i$ or $a_i \pmod{q-1}$ and $b_{i+1} \equiv b_i$, $2b_i$ or $b_i + 1 \pmod{q-1}$ depending on which set S_1, S_2 or S_3 contains x_{i-1}. Making use of Floyd's cyclic algorithm, Pollard computes the six-tuple $(x_i, a_i, b_i, x_{2i}, a_{2i}, b_{2i})$ until $x_i = x_{2i}$. At this stage we have $\beta^r = \alpha^s$ with $r \equiv a_i - a_{2i}$ and $s \equiv b_i - b_{2i} \pmod{q-1}$. It follows that $r\,\mathrm{DLog}_\alpha(\beta) \equiv s \pmod{q-1}$. Since there are only $d = \mathrm{GCD}(r, q-1)$ possible values for $\mathrm{DLog}_\alpha \beta$, if d is small then each of these possibilities can be list to obtain the correct value.

Making the heuristic assumption that the sequence $\{x_i\}$ is a random sequence in $\mathbb{F}_q^*$, then the expected time complexity of this method is $O(\sqrt{q-1})$ field operations.

Pohlig-Hellman method

The Pohlig-Hellman method computes discrete logarithms in finite fields taking advantage of the integer factorization of $q-1$. Let $q-1 = \prod_{i=1}^{t} p_i^{e_i}$ where p_i is a prime number and e_i is a positive integer, for $1 \leq i \leq t$. If $x = \mathrm{DLog}_\alpha(\beta)$ then the approach of Pohlig and Hellman is to obtain x modulo $p_i^{e_i}$ for each i and then use the Chinese Remainder Theorem to get x modulo $q-1$. Then we can reduce to consider the case where $q-1 = p^e$ with p a prime number and

e a positive integer. Let $e_0 = \lceil e/2 \rceil$ and $e_1 = \lfloor e/2 \rfloor$. We have that $x = x_0 + p^{e_0}x_1$, where $0 \leq x_0 < p^{e_0}$ and $0 \leq x_1 < p^{e_1}$. So $\beta = \alpha^x = \alpha^{x_0+p^{e_0}x_1}$. Elevating both sides to p^{e_1} we get

$$\beta^{p^{e_1}} = \alpha^{p^{e_1}x_0+p^{e_0}p^{e_1}x_1} = \alpha^{p^{e_1}x_0+p^{e}x_1} = \alpha^{p^{e_1}x_0},$$

where the last equality holds because α is a primitive element in $\mathbb{F}_q$. Then $x_0 = \mathrm{DLog}_{\alpha}^{p^{e_1}}(\beta^{p^{e_1}})$. Since $\beta\alpha^{-x_0} = \alpha^{p^{e_0}x_1}$, then $x_1 = \mathrm{DLog}_{\alpha}^{p^{e_0}}(\beta\alpha^{-x_0})$. Using one of the two previous methods we determine x_0 and x_1, and thus x.

For the case where $q - 1 = p^e$ this technique requires $O(e(\log(q-1) + \sqrt(p)\log(p)))$ [PH78].

Then, using Pohlig-Hellman algorithm in conjunction with Baby-step Giant-step method or with Pollard-ρ method one can compute discrete logarithms in $\mathbb{F}_q$ in approximately $O(\sqrt{I})$ field operations, where I is the largest prime dividing $q - 1$. Note that in this case the characteristic p of the field is irrelevant. Basing on the computationally capacity of modern computers is reasonable to use this method when I is relatively small, say less than 2^{36}. As a consequence, if one is going to design a cryptographic system based on a finite field $\mathbb{F}_q$ one must select q such that the factorization of $q - 1$ contains a suitable large prime.

Index Calculus

Unlike the Shanks and Pollard-ρ methods, which take exponential time ($O(\sqrt{q-1})$ field operations), index calculus techniques are subexponential with heuristic running time $L[q, 1/3, c]$, and even $L[q, 1/4, c]$ in some cases. The basic idea of index calculus algorithms dates back in 1922 in the work of Kraitchik [Kra22, pp.119-123].

Let $\mathbb{F}_q$ be the finite field in which we want to compute the discrete logarithm and let α be a primitive element of $\mathbb{F}_q$. Consider a subset $S = \{p_1, p_2, \ldots, p_t\}$ of $\mathbb{F}_q$ with the property that a "significant " fraction of all elements in $\mathbb{F}_q$ can be written as the product of elements from S. The set S is usually called the *factor base* for the index calculus method. The index calculus method consists of two main stages. In the first stage one attempts to find the discrete logarithms of all the elements of S. In the second stage one compute logarithms of elements in $\mathbb{F}_q$ which are not in S. To obtain discrete logarithms of elements of S we proceed as follows. We pick a random integer a and attempt to write α^a as a product of elements in S, $\alpha^a = \prod_{i=1}^{t} p_i^{\lambda_i}$. If we are successful then we have that

$$a \equiv \sum \lambda_i \,\mathrm{DLog}_{\alpha} p_i \pmod{q-1}.$$

After collecting a sufficient number, i.e greater than t, of relations of this type, the corresponding system of equations can be expected to have a unique solution for the variables $\mathrm{DLog}_{\alpha} p_i$, $1 \leq i \leq t$. Let $\beta \in \mathbb{F}_q^*$. To compute $\mathrm{DLog}_{\alpha}\beta$ we repeatedly pick an integer s at random until $\alpha^s\beta$ can be written as the product of elements in S, that is $\alpha^s\beta = \sum_{i=1}^{t} p_i^{b_i}$. From this relation

we have that

$$\mathrm{DLog}_\alpha \beta \equiv \sum_{i=1}^{t} b_i \,\mathrm{DLog}_\alpha p_i - s \pmod{q-1}.$$

Note that to complete the description of the method one should precise how to choose the set S and how to efficiently generate the relations expressing an element of $\mathbb{F}_q$ as product of elements in S.

In the last fifty years several different index calculus algorithms were developed for $\mathbb{F}_q$ with $q = p^n$, and the best algorithms vary with the relative sizes of the characteristic p and the extension degree n.

For a long time the best index calculus algorithms for discrete logs had running times of the form $L[q, 1/2, c]$ for various constants $c > 0$. The first practical method that broke through this running time barrier was Coppersmith's algorithm [Cop84] for discrete logs in fields of size $q = p^n$ where p is a small prime and n is large. It had running time of approximately $L[q, 1/3, c]$ with c varying slightly with the values of p and n. This initial progress quickly led to the introduction of several other heuristic algorithms with similar running time. For a long time, index calculus algorithms focused on field with small characteristic, prime fields and occasionally fields of the form $\mathbb{F}_{p^n}$ for small values of n [Sch00]. The view changed in 2006, when was showed that taken together, the Number Field Sieve [JL06] and the Function Field Sieve [AH99] are enough to cover the whole range of finite fields. Both these algorithms have heuristic running time of $L[q, 1/3, c]$. Essentially, the consequence was to split the finite field in three groups, small characteristic with complexity $L[q, 1/3, (32/9)^{1/3}]$, medium characteristic with complexity $L[1/3, (128/9)^{1/3}]$ and large characteristic with complexity $L(1/3, (64/9)^{1/3})$. In 2013 and 2014, two notable improvements on the complexity of discrete logarithm algorithms have been made: two variants of the Number Field Sieve have been designed for finite fields with medium to high characteristic [BP14] and the complexity for finite fields with small characteristic have been dropped to approximately $L[q, 1/4, c]$ [Jou14, BGJT14, GKZ14]].

We conclude this subsection by noting that the expected running times of all the previous algorithms are heuristic, and that the best algorithms with rigorously proved expected running time cost approximately $L[p, 1/2, \sqrt{2}]$ [MBG+13].

2.5.2 Zech Logarithm Table

In small finite fields is useful to employ tables of Zech logarithms to efficiently perform additions and multiplications. The procedure is the following.

Let $\mathbb{F}_q$ be a finite field. Represent the non-zero elements of $\mathbb{F}_q$ as the powers of a primitive element $\alpha \in \mathbb{F}_q$. Then, identifying the elements of $\mathbb{F}_q$ with their discrete logarithms, the multiplication of two elements in $\mathbb{F}_q$ is reduced to the addition of the corresponding two discrete logarithms, since

$$\mathrm{DLog}_\alpha(\beta\gamma) = \mathrm{DLog}_\alpha(\beta) + \mathrm{DLog}_\alpha(\gamma),$$

where the addition is modulo $q-1$. In this representation, to perform the addition of two elements $\beta, \gamma \in \mathbb{F}_q$ one needs to compute the discrete logarithm $\mathrm{DLog}_\alpha(\beta+\gamma)$. Since $\alpha^a+\alpha^b = \alpha^a(1+\alpha^{b-a})$, to do that it suffices to compute the discrete logarithms for sums involving 1, i.e. logarithms of the form $\mathrm{DLog}_\alpha(1+\gamma)$. This motivates the following definition

Definition 2.5.3 (Zech's Logarithm)**.** Let $\mathbb{F}_q$ be a finite field and let α be a primitive element of $\mathbb{F}_q$. The *Zech's logarithm* (or *Jacobi logarithm*) with base α of an integer n is the integer $\mathrm{Z}_\alpha(n)$ defined by the equation

$$1+\alpha^n = \alpha^{\mathrm{Z}_\alpha(n)},$$

where the case $\alpha^n = -1$ is excluded.

When q is small, say less than 2^{20}, the Zech logarithms can be precomputed and, when needed for an addition, recovered by a simple table lookup. Moreover, exploiting some properties of Zech's logarithm [Hub90] one can reduce the storage requirements from $q = p^k$ to roughly $q/6k$.
It is clear that a table lookup of Zech logarithms becomes impractical for finite fields with large size. In this case is preferable to use a polynomial representation.

Decoding Problem for Cyclic Codes

The main problem of information theory can be described as follows. Suppose that a source of data has been transmitted over a noisy channel. In this context, the following question arise naturally: how can we tell when the original data has been changed? And when it has, how can we recover the original data? One of the goal of Coding Theory is to answer to the two previous questions.

In the first section of this chapter we formalize the above problems and introduce error correcting codes. In Section 3.2 we recall some basic properties of linear codes and discuss the complexity of decoding problem for these codes. Finally, in Section 3.3, after recalling some properties of cyclic codes, we address the decoding problem for cyclic codes and discuss interesting open problems regarding general error locators.

Most of the results in this chapter are well-known in the literature. Here we use as main references [MS77], [PW72] and [PHB98].

3.1 An overview on error correcting codes

Coding theory is the study of the properties of codes and their suitability for a specific application. One of these applications is the design of efficient and reliable data transmission methods. A code designed with this purpose is called an *error correcting code*. There are two fundamentally different types of codes: *block codes* which break the sequence of information digits into k-symbol blocks, and *tree codes* which operates on the information sequence without breaking it into independent blocks.

In this thesis attention is focused on block codes. From now on, when we say error correcting codes we actually mean block error correcting codes.

Definition 3.1.1. Let $\mathbb{F}_q$ be a finite field. A subset C of $\mathbb{F}_q^n$ of size M is called a *block code* (or also a q-ary block code) of length n and size M over $\mathbb{F}_q$, and its elements are called *codewords*. We call an element of $\mathbb{F}_q$ a *symbol*, and an element of $\mathbb{F}_q^n$ (which is not necessary in C) a *word*.

A general communication system which uses an error correcting code C is showed in Figure 3.1. It consists of essentially five parts:

1. An *information source*: which produces a message $m = (m_1, \ldots, m_k) \in \mathbb{F}_q^k$ to be communicated

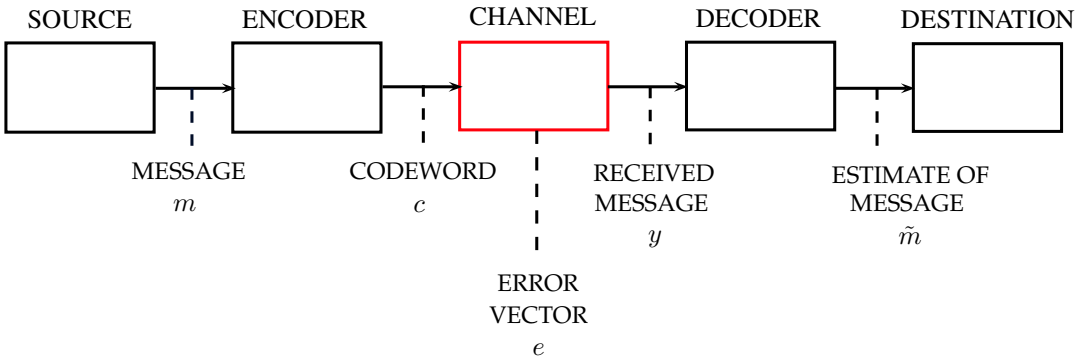

Figure 3.1: A simple model of communication system. We omit to represent both the transmitter and the receiver.

2. An *encoder*: which produces a codeword $c = (c_1, \ldots, c_n) \in C$,where $n > k$, from a message m
3. A *transmitter*: which operates on the codeword c in order to produce a signal suitable for transmission over the channel
4. A *noise channel*: which is the medium used to transmit the signal from a transmitter to a receiver, and it is supposed to be noisy
5. A *receiver*: performing the inverse operation of the transmitter, i.e. recovering the word $y \in \mathbb{F}_q^n$ corresponding to the received signal
6. A *decoder*: which tries to recover the original message m from y
7. A *destination*: the person who the message is addressed to.

We call *encoding* the process of converting the message $m \in \mathbb{F}_q^k$ into a codeword c of a given code C. The process of recovering the messages m from the received word y is said *decoding*.

The central idea in using error-correcting codes for communication data is that the sender encodes the message in a redundant way by using an error-correcting code in a such way (depending on the channel used for the transmission) that the redundancy allows the receiver to detect and/or correct possible transmission errors.

Example 3.1.2. Let us consider the set

$$C = \{00000, 01011, 10101, 11110\}.$$

C is a binary code of length $n = 5$ with $M = 4$ codewords.
Suppose that we want to employ C in order to guarantee a reliable transmission data over a noisy channel, and that the (corrupted) codeword $y = 00101$ is received. The set of possible errors corresponding to the word y is given by $E = \{00101, 01110, 10000, 11010\}$.

From the previous example it is clear that to decide which error out of a set of errors is the "right one", that is to decode, one needs to make additional assumptions about the likelihood of errors and error patterns. One need also to specify how the sent codewords c and the error patterns e are combined to form the received codewords $y = f(c, e)$.
Throughout all the thesis we will assume that for a q-ary code, codewords c and error patterns e are combined through a modulo q addition, that is $y = c + e$ where $+$ is the addition in $\mathbb{F}_q^n$. A such channel is called an *addition channel*. We will also assume that the channel is an q-ary *symmetric channel*. In the next definition we recall the notion of symmetric channel.

Definition 3.1.3 (q-ary symmetric channel)**.** A *q-ary symmetric channel* (qSC) *without memory* is a discrete channel with q-ary symbols both as inputs and as outputs. Moreover it satisfies the following properties

- the probability that a symbol changes into another is the same for all the $q-1$ symbols
- the probability that a symbol changes does not depend on its position in the transmitted word
- if the ith component of a transmitted word changes, then this fact does not affect the probability that its jth component changes

The transition probabilities of the q-ary symmetric channel depend on a single parameter ϵ as follows

$$\mathbb{P}[b|a] = \begin{cases} 1-\epsilon & b = a \\ \epsilon/(q-1) & b \neq a \end{cases}$$

where $\mathbb{P}[b|a]$ denotes the probability to receive the symbol b given that the symbol a was sent. In other words, in a q-ary symmetric channel every symbol is left untouched with probability $1-\epsilon$ and is distorted to each of the $q-1$ possible different symbols with probability ϵ.

An important requirement in order to define a transmission model is to fix a distance measure between codewords. The most prevalent measure for our purpose is Hamming distance, which is defined as follows.

Definition 3.1.4. Given two vectors $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{n-1})$ in $\mathbb{F}_q^n$, we define the *Hamming distance* between x and y as the number of components in which they differ

$$\mathrm{d}(x, y) = |\{0 \leq i \leq n-1 \mid x_i \neq y_i\}|.$$

The *Hamming weight* of a vector $x \in \mathbb{F}_q^n$ is the number, $\mathrm{w}(x)$, of its non-zero components, i.e. $\mathrm{w}(x) = \mathrm{d}(x, 0)$.

Let C be a q-ary code, $y \in \mathbb{F}_q^n$ be the received word, and e be the error vector occurred in the transmission. So, $y = x + e$, where x is the transmitted codeword.

Definition 3.1.5. We say that C has *error correction capability* t (or that C can correct up to t errors) if for any error vector e with $\mathrm{w}(e) \leq t$, C allows to recover x. Similarly, we say that C has *error detection capability* s (or that C can detect up to s errors) if for any error vector e with $\mathrm{w}(e) \leq s$, C allows to detect that an error occurred in the transmission.

Next two definitions introduce two important parameters of an error correcting code.

Definition 3.1.6. The *rate* R of a q-ary block code of length n with M codewords is given by

$$R = (\log_q M)/n$$

The rate of a code is a measure of the efficiency of the code.

Definition 3.1.7. The *redundancy* r of a q-ary block code of length n with M codewords is given by

$$r = n - \log_q M.$$

We can say that the birth of the subject of coding theory for data transmission occurred in 1948 when Shannon published the paper " A Mathematical Theory of Communication" [Sha48]. His work focuses on the problem of how best to encode the information a sender wants to transmit. He established the limits of the gains possible with coding theory, and proved the existence of codes that could effectively reach these limits. More precisely, Shannon showed the following theorem. Beforing stating it, we introduce the capacity of a q-ary symmetric channel.

Definition 3.1.8. Given a q-ary symmetric channel with probability of symbol error ϵ, we say that the capacity of the channel is

$$C(\epsilon) = 1 + \epsilon \log_q(\epsilon) + (1-\epsilon)\log_q(1-\epsilon) - p\log_q(q-1).$$

Theorem 3.1.9 (Shannon)**.** *For any $\delta > 0$ and $R < C(\epsilon)$, there is a (linear) code over $\mathbb{F}_q$ of rate $R' \geq R$ with $P_{err} < \delta$, where P_{err} is the probability that the output of the decoder does not correspond to the originally transmitted vector.*

So, if one wishes to communicate over a channel of capacity Q at a rate R, then one can do so as reliably as desired, if and only if $R < Q$. Although Shannon [Sha48] answered to the question "Do good codes exist?" affirmatively, his work raises two other questions " How can we construct such codes?" and "How can we decode them?". In a sense, the study of error correction codes is all about these two questions.

3.2 Linear Codes

3.2.1 Basic definitions

In this section we summarize basic definitions regarding linear block codes.

Definition 3.2.1. Let C be a q-ary block code of length n. We say that C is a linear (block) code if it is a linear subspace of the vector space $\mathbb{F}_q^n$. If the dimension of the subspace C is k, then we say that the code C has *dimension* k and that C is a $[n, k]$-code.

We do not treat in this thesis the case of non-linear block codes, so often we will use the word "code" for referring to linear block code. When we do not specify the field, we implicitly mean that the code is defined over $\mathbb{F}_q$. Note that if C is an $[n, k]$-code over $\mathbb{F}_q$, then $|C| = q^k$ where $|C|$ denotes the cardinality of C.

As subspace of $\mathbb{F}_q^n$, an $[n, k]$-code admits a basis. This leads to the definition of a generator matrix of a code.

Definition 3.2.2. Let C be an $[n, k]$-code over $\mathbb{F}_q$. Any matrix G whose rows form a basis for C as a k-dimensional subspace of $\mathbb{F}_q^n$ is called a generator matrix of C.

If G has the form $G = [\mathbb{1}_k|A]$, where $\mathbb{1}_k$ is the $k \times k$ identity matrix, then G is called a generator matrix *in standard form*. In general, there are many generator matrices for a codes, nevertheless any code has a unique generator matrix in standard form.

Given $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{n-1})$ in $\mathbb{F}_q^n$, we denote by $x \cdot y$ the scalar product of x and y in $\mathbb{F}_q^n$, that is $x \cdot y = \sum_{i=0}^{n-1} x_i y_i$. We recall that x and y are said *orthogonal* if $x \cdot y = 0$.

Definition 3.2.3. Let C be an $[n, k]$-code. The set $C^\perp$ of vectors in $\mathbb{F}_q^n$ which are orthogonal to all codewords of C is again a linear code and it is called the *dual code* of C.

We note that the dual code of an $[n, k]$-code C is an $[n, n - k]$-code.

Definition 3.2.4. A *parity-check matrix* H for an $[n, k]$-code is a generator matrix of $C^\perp$.

By previous definitions we easily get that if C is an $[n, k]$-code then a generator matrix G of C has size $k \times n$ and a parity-check matrix H has size $(n - k) \times n$. To check if an n-vector $x \in \mathbb{F}_q^n$ belongs to C it is sufficient to compute Hx^T. Indeed it holds that

$$\forall x \in \mathbb{F}_q^n,\ Hx^T = 0 \longleftrightarrow x \in C \tag{3.1}$$

Definition 3.2.5. Let C be an $[n, k]$-code over $\mathbb{F}_{q^l}$. The *subfield subcode* $C\restriction_{\mathbb{F}_q}$ of C with respect to $\mathbb{F}_q$ is the set of codewords in C each of whose components is in $\mathbb{F}_q$.

Because C is linear over $\mathbb{F}_{q^l}$, then $C\restriction_{\mathbb{F}_q}$ is a linear code over $\mathbb{F}_q$.

Remark 3.2.6. Let C be an $[n, k]$-code over $\mathbb{F}_{q^l}$ and H be a parity-check matrix of C. To find a parity check matrix for $C\restriction_{\mathbb{F}_q}$ considering a basis $\{b_1, b_2, \ldots, b_l\} \subset \mathbb{F}_{q^l}$ of $\mathbb{F}_{q^l}$ over $\mathbb{F}_q$. Each element $z \in \mathbb{F}_{q^l}$ can be uniquely written as $z = z_1 b_1 + \cdots + z_l b_l$, where $z_i \in \mathbb{F}_q$ for $1 \leq i \leq l$. Associate to z the column vector $\bar{z} = (z_1, \ldots, z_l)^T$. Create $\bar{H}$ from H by replacing each entry h by the vector $\bar{h}$. Because H is an $(n - k) \times n$ matrix with entries in $\mathbb{F}_{q^l}$, $\bar{H}$ is a $l(n - k) \times n$ matrix over $\mathbb{F}_q$. A parity check matrix for $C\restriction_{\mathbb{F}_q}$ is obtained from $\bar{H}$ by deleting dependent rows. We denote this parity check matrix by $H\restriction_{\mathbb{F}_q}$.

Definition 3.2.7. The *minimum distance* (or simply the *distance*) of a code C is the smallest (Hamming) distance between distinct codewords, that is

$$\mathrm{d}(C) = \min\{\mathrm{d}(x, y) \mid x, y \in C, x \neq y\}.$$

If C is a $[n, k]$-code and we known the distance of C then we also refer to it as an $[n, k, d(C)]$-code.

The following result shows that it is possible to define the distance of a linear code using the weight of codewords instead of mutual distance between all codewords.

Proposition 3.2.8. *Let C be an $[n, k, d]$-code. Then $d(C) = \min\{\mathrm{w}(c) \mid c \in C, c \neq 0\}$.*

Let C be an $[n, k, d]$ linear code over $\mathbb{F}_q$. If D is a vector subspace of C, then we say that D is a (linear) subcode of C. Moreover we have $\mathrm{d}(C) \leq \mathrm{d}(D)$.

Definition 3.2.9. Let C be an $[n, k, d]$-code. We denote by A_i the number of the codewords of weight i. The set of $\{A_i\}_{0 \leq i \leq n}$ is called the weight distribution of C.

Definition 3.2.10. Let C_1 and C_2 be two linear codes. We say that C_1 and C_2 are *permutation equivalent* if there is a permutation of coordinates which sends C_1 to C_2.

Two permutation equivalent codes have the same weight distribution.

Proposition 3.2.11. *Let C be linear code and let H be a parity-check matrix of C. Then C has distance d if and only if H has a set of d linearly dependent columns and any set of $d - 1$ columns is linearly independent.*

The following theorem is a consequence of the previous result. It gives an upper bound for the distance of a code.

Theorem 3.2.12 (Singleton bound)**.** *Let C be an $[n, k, d]$-code. Then*

$$d \leq n - k + 1$$

A code reaching the equality in the Singleton bound is called a *maximum distance separable code* or an *MDS* code.

Definition 3.2.13. Let x be any vector in $\mathbb{F}_q^n$ and let C be an $[n, k, d]$ code with parity-check matrix H. The vector $s \in (\mathbb{F}_q)^{n-k}$ such that $s = Hx^T$ is called the *syndrome vector of x with respect to H* (or simply the *syndrome of x*).

Equation (3.1) tell us that a vector $x \in \mathbb{F}_q^n$ is a codeword of C if and only if the syndrome of x is zero. The syndrome, however, tell us more than a vector being in the code or not. Suppose the vector c was transmitted and v was received, where $v = c + e$, with e the error vector. Note that

$$s = Hv^T = H(c + e)^T = Hc^T + He^T = He^T,$$

since $c \in C$. Then, the syndrome does not depend on the received vector but on the error vector. But we can say something more.

Proposition 3.2.14. *Let C be an $[n, k, d]$-code and let H be a parity-check matrix of C. Then, there is a* 1-1 *correspondence between errors of weight $\leq \lfloor (d-1)/2 \rfloor$ and syndromes.*

This property of linear codes justify the following definition

Definition 3.2.15. Let C be an $[n, k, d]$-code over $\mathbb{F}_q$. We call *correctable syndromes* the syndrome vectors $s \in \mathbb{F}_q^{n-k}$ corresponding to errors of weight $\mu \leq t$ with $t = \lfloor (d-1)/2 \rfloor$.

Proposition 3.2.16. *Let C be an $[n, k, d]$-code over $\mathbb{F}_q$. Then*

- *C has detection capability $d - 1$*
- *C has error correction capability $t = \lfloor d - 1 \rfloor / 2$.*

The previous result shows that the distance of a code reflects the error correction capability of the code. Evidently, we would like t, and hence also d, to be large.

We conclude this section summarizing the properties a good error correcting code should have. In the study of error correcting codes it is important to construct $[n, k, d]$-codes having the following properties:

1. large rate R;
2. large distance d;
3. admitting efficient encoding and decoding algorithms.

There is a trade-off between the rate R and the minimum distance d for any linear code. Indeed, by the Singleton bound and the definition of rate of a code, we have that

$$R + \frac{d}{n} \leq 1 + \frac{1}{n}.$$

So we have to make a compromise between the rate and the minimum distance. The third property above is also very important for a code, as codes with large rate and distance may be useless if they have no efficient encoding and decoding algorithms. In the next section we trait the problem of decoding linear codes.

3.2.2 Decoding Linear Codes

Suppose to use an $[n, k]$-code C over $\mathbb{F}_q$ for a reliable transmission of data over a noisy channel, and to receive the word $y \in \mathbb{F}_q^n$. The optimal decoding strategy is to find the codeword $c \in C$ that maximizes the probability $Pr(c|y)$ that c was transmitted given that y was received. On a q-ary symmetric channel, this strategy is equivalent to find the closest codeword to y in the Hamming metric [MS77].

Definition 3.2.17. A decoder for C that always finds the closest codeword to r (or one of the closest codewords) in the Hamming metric is said to be a *maximum-likelihood* decoder.

Maximum-likelihood decoding of a q-ary $[n, k, d]$-code can be trivially accomplished in $O(q^k)$ by simply computing the Hamming distance between the received word y and all the q^k codeword of C. However, this procedure is computationally infeasible when the number of codewords is large. An other method accomplishing this task is the following, which costs $O(q^{n-k})$. Let C be an $[n, k]$-code over $\mathbb{F}_q$. For any vector $u \in \mathbb{F}_q^n$ the set

$$u + C = \{u + c \mid c \in C\}$$

is called a coset of C. Every vector of $\mathbb{F}_q^n$ is contained in such a coset. Since C is a linear subspace of $\mathbb{F}_q^n$, two cosets are either disjoint or identical. Suppose that a codeword x was sent via a noisy channel, and a vector y is received, then the error vector is $e = y - x$. Let H denote a parity check matrix of C. It follows that $He^T = Hy^T$, that is the syndrome of e corresponds to the syndrome of y. Obviously, all the vectors of a coset have the same syndrome. The minimum weight vector of a coset is called the *coset leader* (if there is more than one vector with the same minimum weight, choose one at random and take it as the coset leader). A naive decoding procedure consists in first computing the syndrome of a received vector Hy^T, then take the coset leader e of the coset having syndrome Hy^T as the error vector, and finally decode the received vector y to be $y - e$. Since the number of possible syndromes is q^{n-k}, this method costs $O(q^{n-k})$.

Although maximum-likelihood decoding is the optimal decoding strategy, Berlakamp et al. [BMvT78] showed that for the general class of linear codes, polynomial-time maximum-likelihood decoding algorithms are unlikely to exist (unless P=NP). Bruck and Naor [BN90] showed that this remains true even if the code is known in advance and can be preprocessed for a long as desired in order to devise a decoding algorithm. Moreover, such algorithms are not known today even for any specific family of useful codes, such as the binary BCH codes, for instance. On the contrary, in [GV05], it was proved the NP-completeness of maximum likelihood decoding for generalized Reed-Solomon codes. A potential ways to circumvent these results is to attempt to correct only a limited number of errors. A decoding strategy which is sub-optimal is the following.

Definition 3.2.18 (Bounded-distance Decoding)**.** A decoder is said to be *bounded-distance* if there exists an integer $t > 0$ such that the decoder always finds the closest codeword c to a channel output r, provided that $d(c, r) \leq t$.

One would hope for a result of the form: "There exists an $\epsilon > 0$, such that for every $[n, k, d]$-code C, the bounded distance decoding problem for C with $t = \epsilon d$ is solvable in polynomial time". It is still not known whether bounded-distance decoding is NP-hard for the general class of linear codes. In [Var97] Vardy conjectures that this is so and that the NP-hardness of the problem of computing the minimum distance of a linear code should be instrumental in trying to prove this conjecture. However, for many algebraic families of codes, such as BCH codes and most algebraic-geometry codes, we know polynomial-time bounded-distance

decoding algorithms that achieve the error-correction radius of the code $t = \lfloor (d-1)/2 \rfloor$. In the next section we shall show the decoding of some important class of linear cyclic codes.

3.3 Cyclic Codes

3.3.1 Basic definitions

An important class of linear codes is cyclic codes. They are very used in the applications since they can be implemented fairly simply (for instance using LFSRs) and their mathematical structure is reasonable well understood.

Definition 3.3.1. A code C is *cyclic* if it is invariant under any cyclic right shift of the coordinates, i.e.

$$(c_0, \ldots, c_{n-1}) \in C \rightarrow (c_{n-1}, c_0 \ldots, c_{n-2}) \in C$$

Example 3.3.2. Let $C = \{000, 110, 101, 011\}$. C is an example of binary cyclic code.

A useful way to describe algebraic properties of cyclic codes is to represent its codewords as polynomials. Let $(x^n - 1)$ be the ideal generated by $x^n - 1 \in \mathbb{F}_q[x]$. Then the elements of the quotient ring $R_n = \mathbb{F}_q[x]/(x^n - 1)$ can be represented by polynomials of degree less than n and this quotient ring is isomorphic to $\mathbb{F}_q^n$ as vector spaces over $\mathbb{F}_q$ by the correspondence

$$v_0 + v_1 x + \cdots + v_{n-1} x^{n-1} \in R_n \longleftrightarrow (v_0, v_1 \ldots, v_{n-1}) \in \mathbb{F}_q^n \tag{3.2}$$

Because of this isomorphism, for a word $v \in \mathbb{F}_q^n$ we use interchangeably the vector notation or the polynomial notation $v(x)$. In particular we will say that $v(x)$ has weight w if $\mathrm{w}(v) = w$.

Thanks to the characterization 3.2, we can see linear codes of length n as subsets of R_n. The following theorem states that cyclic codes of length n over $\mathbb{F}_q$ correspond to ideals in R_n.

Theorem 3.3.3. *Let C be an $[n, k, d]$-code over $\mathbb{F}_q$. Then C is cyclic if and only if C is an ideal of R_n.*

Since R_n is a principal ideal ring, any ideal C of R_n is generated by a polynomial $g(x) \in R_n$, i.e. $C = (g(x))$. If we require that $g(x)$ is monic and of lowest degree, then it is unique. Such polynomial g is called the *generator polynomial* of the code C. Since ideals of R_n are of the form $J/(x^n - 1)$ with J an ideal of $\mathbb{F}_q[x]$ containing $(x^n - 1)$, we also have that cyclic codes of length n over $\mathbb{F}_q$ are generated by divisors of $(x^n - 1)$ in $\mathbb{F}_q[x]$.

The next theorem show some other elementary properties of cyclic codes.

Theorem 3.3.4. *Let C be a cyclic code of length n over $\mathbb{F}_q$, and let $g(x)$ its generator polynomial. Then*

1. *If the dimension of C is k, then $\deg(g) = n - k$.*

2. *If* $g(x) = g_0 + g_1x + \cdots + g_{n-k}x^{n-k}$*, then a generator matrix for* C *is given by*

$$G = \begin{pmatrix} g_0 & g_1 & \dots & g_{n-k} & 0 & \dots & 0 \\ 0 & g_0 & \dots & g_{n-k-1} & g_{n-k} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & g_0 & g_1 & \dots & g_{n-k} \end{pmatrix}.$$

Definition 3.3.5. Let C be a cyclic code of length n over $\mathbb{F}_q$. We say that C is a *primitive* cyclic code if $n = q^l - 1$ for some positive integer l.

Definition 3.3.6. Let C be a cyclic code. A linear subcode C' of C that is cyclic will be called a *cyclic subcode*. In this case we will write $C' \subset C$ if C' is not zero.

Proposition 3.3.7. *Let* C_1 *and* C_2 *be cyclic codes over* $\mathbb{F}_q$ *with generator polynomial* $g_1(x)$ *and* $g_2(x)$ *respectively. Then* $C_1 \subset C_2$ *if and only if* $g_2(x) \mid g_1(x)$.

From now on we impose the restriction that $\mathrm{GCD}(n, q) = 1$ (for the case $\mathrm{GCD}(n, q) > 1$ see [vL91] and [CMSvS91]). By Proposition 2.3.14, this guarantees that the polynomial $x^n - 1 \in \mathbb{F}_q[x]$ has distinct roots in its splitting field and this leads to a very useful characterization of a cyclic code from the roots of its generator polynomial.

Definition 3.3.8. Let C be a cyclic code of length n over $\mathbb{F}_q$ generated by the polynomial $g(x)$. Let $\mathbb{F}$ be the splitting field of $x^n - 1$ over $\mathbb{F}_q$ and let α be a primitive nth root of unity in $\mathbb{F}$. The set

$$\tilde{S}_{C,\alpha} = \{0 \leq i \leq n-1 \mid g(\alpha^i) = 0\}.$$

is called the *complete defining set of* C w.r.t. α. Also, the roots of unity $\{\alpha^i \mid i \in \tilde{S}_{C,\alpha}\}$ are called the *zeros* of the cyclic code C.

Note that, given a cyclic code C, different choices of α give different complete defining sets of C. As we will see in a moment, actually we do not care about the choice of α.

Fix a primitive nth root of unity α in the splitting field of $x^n - 1$ over $\mathbb{F}_q$. Then a cyclic code of length n over $\mathbb{F}_q$ is defined by its complete defining set w.r.t. α. In fact,

$$c \in C \longleftrightarrow c(\alpha^i) = 0 \text{ for any } i \in \tilde{S}_{C,\alpha}. \tag{3.3}$$

We observe that this characterization of cyclic codes is not true if we drop the assumption that $\mathrm{GCD}(n, q) = 1$.

Definition 3.3.9. Let C_1 and C_2 be two cyclic codes of length n over $\mathbb{F}_q$. We say that C_1 and C_2 are *naturally equivalent* if there are two nth roots of unity in the splitting field of $x^n - 1$ over $\mathbb{F}_q$, α and β, s.t.

$$S_{C_1,\alpha} = S_{C_2,\beta}.$$

We have that two naturally equivalent cyclic codes are also permutation equivalent. The converse, in general, is not true.

Theorem 3.3.10. *Let C_1 and C_2 be naturally equivalent cyclic codes. Then*

$$\mathrm{d}(C_1) = \mathrm{d}(C_2).$$

Moreover, if C_1 is a cyclic code of length n over $\mathbb{F}_q$ and α and β are primitive nth roots of unity in the splitting field of $x^n - 1$ over $\mathbb{F}_q$, then there is a unique cyclic code C_2 of the same length over $\mathbb{F}_q$ s.t. $S_{C_1,\alpha} = S_{C_2,\beta}$.

Therefore, different choices of α define the same cyclic code up to equivalence. From now one we denote the complete defining set of C w.r.t. α by $\tilde{S}_C$ instead of $\tilde{S}_{C,\alpha}$ and we refer to it as the complete defining set of C.

By (3.3), if $\tilde{S}_C = \{i_1, i_2, \ldots, i_{n-k}\}$ is the complete defining set of a cyclic code C of length n and dimension k, then a parity-check matrix of C is given by

$$H = \begin{pmatrix} 1 & \alpha^{i_1} & \alpha^{2i_1} & \ldots & \alpha^{(n-1)i_1} \\ 1 & \alpha^{i_2} & \alpha^{2i_2} & \ldots & \alpha^{(n-1)i_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{i_{n-k}} & \alpha^{2i_{n-k}} & \ldots & \alpha^{(n-1)i_{n-k}} \end{pmatrix}, \tag{3.4}$$

because

$$Hc = \begin{pmatrix} c(\alpha^{i_1}) \\ c(\alpha^{i_2}) \\ \vdots \\ c(\alpha^{i_{n-k}}) \end{pmatrix}.$$

We note that the entries of matrix H in (3.4) are in $\mathbb{F}$ rather than in $\mathbb{F}_q$. A parity-check matrix for C with entries in $\mathbb{F}_q$ is given by $H\restriction_{\mathbb{F}_q}$ (see Remark 3.2.6).

Since $g(\alpha^i) = 0$ implies that also $g(\alpha^{qi}) = g(\alpha^i)^q = 0$, we have that the complete defining set of C is partitioned into cyclotomic classes. More precisely we have that for some $s \geq 1$,

$$S_C = \bigsqcup_{j=1}^{s} C_{i_j},$$

where C_{i_j} is the q-cyclotomic coset modulo n over $\mathbb{F}_q$ containing the integer i_j. This means that any subset of $\tilde{S}_C$ containing at least one element per cyclotomic class is sufficient to specify the code C. We call such a set a *defining set of* C. A defining set of C having only one representative element per cyclotomic class is called a *base set of* C. We will use S_C to denote a defining set which is not necessarily a complete defining set. As for $\tilde{S}_C$, we have that

$$c \in C \longleftrightarrow c(\alpha^i) = 0 \text{ for any } i \in S_C. \tag{3.5}$$

There are several known lower bounds for the distance of a cyclic code. The oldest of these is the BCH bound [BRC60].

Theorem 3.3.11 (BCH Bound)**.** *Let C be a cyclic code of length n over $\mathbb{F}_q$ and let S its complete defining set. Suppose C has distance d. If S contains $\delta-1$ consecutive elements for some integer δ, then $d \geq \delta$.*

We close this section introducing three important families of cyclic codes: BCH codes, Reed-Solomon codes and quadratic residue (QR) codes.

BCH codes are cyclic codes designed to take advantage of the BCH Bound in the following sense: by the BCH bound, if we would like to construct a cyclic code C having distance at least δ and dimension as small as possible, we can accomplish this by choosing as complete defining set of C the smallest set being a union of q-cyclotomic cosets C_i with $\delta-1$ consecutive elements.

Definition 3.3.12. Let δ be an integer with $2 \leq \delta \leq n$. A *BCH code* C over $\mathbb{F}_q$ of length n and designed distance δ is a cyclic code with complete defining set

$$S = C_b \cup C_{b+1} \cup \cdots \cup C_{b+\delta-2},$$

for some integer $b > 0$.
When $b = 1$, C is called a *narrow-sense BCH code*. Moreover, C is called a *primitive BCH code* if it is a primitive cyclic code.

By the BCH Bound, we have that

Theorem 3.3.13. *A BCH code with designed distance δ has (minimum) distance at least δ.*

BCH codes are very powerful since for any integer $d > 0$ we can construct a BCH code of minimum distance $\geq d$. Note, however, that to find a BCH code of larger distance we have to increase the length n.
By definition, BCH codes are easily constructible in polynomial time. Moreover, as we will see in the next section, they have also efficient decoding algorithms. It turns out that for short lengths, say up to $n < 128$, BCH codes are among the best cyclic codes known. However, Lin and Weldon [LW67] showed that BCH codes are asymptotically bad, in the following sense.

Definition 3.3.14. We say that a family $\mathcal{C}$ of codes over $\mathbb{F}_q$ is *asymptotically good* if there exists an infinite subset of $[n_i, k_i, d_i]$ codes from this family with with $\lim_{i\to\infty} n_i = \infty$ such that both the code rates k_i/n_i and the relative distances d_i/n_i are bounded away from 0, i.e.

$$\liminf_{i\to\infty} k_i/n_i > 0 \quad \text{and} \quad \liminf_{i\to\infty} d_i/n_i > 0$$

A family of codes is said *asymptotically bad* if no asymptotically good subfamily exists.

It is natural to ask whether or not there is any family of asymptotically good cyclic codes. In the last decades some progress has been made in trying to answer to this question, but it remains still open. Some partial positive results are known for quasicyclic codes [Kas74]. In [Cas89] Castagnoli showed that, if the length n_i goes to infinity with i while having a fixed set of prime

factors, then there is no asymptotically good family of cyclic codes C_i of length n_i. Also, Bazzi and Mitter [BM06] have shown that there exists an asymptotically good family of linear codes which are very close to cyclic codes. Willems and Martinez-Perez [MPW06] have shown that if there exists an asymptotically good family of cyclic codes, then there exists an asymptotically good family of cyclic codes with prime lengths.

Reed–Solomon codes [RS60] can be seen as a subfamily of BCH codes. Because of their burst error-correction capabilities, they are used to improve the reliability in data storage systems. Also, they constitute the primary example of MDS codes.

Definition 3.3.15. A *Reed–Solomon code* C over $\mathbb{F}_q$ is a BCH code of length $n = q - 1$.

The family of quadratic residue codes is a subclass of cyclic codes over prime fields. This family contains some very good codes of small length, which makes it a promising family of codes. An interesting open question is whether it is an asymptotically good family of codes.

Definition 3.3.16. A quadratic residue code over $\mathbb{F}_q$ is a cyclic code of odd prime length p over $\mathbb{F}_q$ with q a prime that is a quadratic residue modulo p, that is, $q \equiv r^2 \pmod{p}$ for some nonzero integer r.

3.3.2 Decoding Cyclic Codes

Cyclic codes are not known to be asymptotically bad as the BCH codes. So it would be important to have efficient decoding algorithms for cyclic codes. Unfortunately, until now, we do not know any efficient bounded-distance decoding algorithm for cyclic codes (up to their actual distance).
In the last fifty years many efficient bounded-distance decoders have been developed for special classes of cyclic codes, e.g. the Berlekamp-Massey (BM) algorithm [Mas69] designed for the BCH codes. The most common approach for finding efficient decoding procedures for the class of cyclic codes consists in trying to extend such decoders.

In the first part of this section we briefly recall the classical approach to decode BCH codes. Then we outline the main strategies that have been proposed in the last decades for decoding cyclic codes. Finally we focus our attention on a particular locator polynomial and we show some of their properties.

Decoding BCH codes

From now on until the end of the chapter, we denotes vectors by bold lower-case letters. Moreover, if $v(x) \in \mathbb{F}_q[x]$ with $\deg v(x) < n$ then the vector corresponding to it by (3.2) is denoted by $\mathbf{v}$.

We illustrate now a general strategy to decode BCH codes consisting of four main steps, which we describe in order and later summarize. While the following procedure can be applied to any BCH code, here for simplicity, we show it only for narrow-sense BCH codes.

Let α be a primitive nth root of unity in $\mathbb{F}_{q^m}$, where m is the order of q modulo n, i.e. $m = \mathrm{ord}_n(q)$. Let C be the (narrow-sense) BCH code over $\mathbb{F}_q$ of length n and designed distance δ with defining set $S_C = \{1, 2, \ldots, \delta - 1\}$. As the minimum distance of C is at least δ, C can correct at least $t = \lfloor (\delta - 1)/2 \rfloor$ errors. Let $c(x), v(x), e(x)$ be, respectively, the transmitted code polynomial, the received polynomial and the error polynomial, then $v(x) = c(x) + e(x)$. Suppose that $\mathbf{e}$ has weight $\mu \leq t$ (such an error is said to be a *correctable error*) and that the errors occur in the unknown coordinates $l_1, l_2, \ldots, l_\mu$ with $0 \leq l_1 < l_2 < \cdots < l_\mu \leq n - 1$. Therefore

$$e(x) = e_{l_1} x^{l_1} + e_{l_2} x^{l_2} + \cdots + e_{l_\mu} x^{l_\mu}, \tag{3.6}$$

for some $e_{l_j} \in \mathbb{F}_q^*$, for $1 \leq j \leq \mu$.

Definition 3.3.17. The integers l_j are called the *error positions* associate to $e(x)$, while the e_{l_j}, for $1 \leq j \leq \mu$, are called the *error values* associate to $e(x)$.

Once we obtain $e(x)$, which amount to find the error positions l_j and the error values e_{l_j}, we can decode the received polynomial as $c(x) = v(x) - e(x)$. Note that in the binary case any error is completely characterized by l_j alone, since in this case all the error values are 1.

Remark 3.3.18. Since α is a primitive nth root on unity in $\mathbb{F}_{q^m}$, we have that

$$\alpha^i = \alpha^j \quad \text{for } i, j \in \{0, 1, \ldots, n-1\} \to i = j$$

Therefore, knowing α^{l_j} uniquely determines the error position l_j.

Definition 3.3.19. The elements α^{l_j}, for $1 \leq j \leq \mu$, are called the *error locations* associate to $e(x)$.

By (3.5), we have that

$$v(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \quad \text{for all } 1 \leq i \leq \delta - 1.$$

Definition 3.3.20. The elements $s_i = e(\alpha^i)$, for $0 \leq i \leq n - 1$, are said *syndromes*. If $\tilde{S}_C$ is the complete defining set of C, then s_i is said a *known syndrome* if $i \in \tilde{S}_C$. Syndromes which are not known syndromes are called *unknown syndromes*. Known syndromes which correspond to a base set of C are called *primary syndromes*.

Note that in particular the syndromes s_i, for $1 \leq i \leq \delta - 1$, are known syndromes.

The first step of the procedure is to compute the syndromes s_i, for $1 \leq i \leq \delta - 1$ of the received polynomial $v(x)$. These syndromes lead to a system of equations involving the unknown error locations and the unknown error values. From (3.6), it holds that

$$s_i = e(\alpha^i) = \sum_{j=1}^{\mu} e_{l_j} (\alpha^i)^{l_j} = \sum_{j=1}^{\mu} e_{l_j} (\alpha^{l_j})^i, \quad \text{for } 1 \leq i \leq \delta - 1. \tag{3.7}$$

To simplify the notation, for $1 \leq j \leq \mu$, let $z_j = \alpha^{l_j}$. By point 2 of Theorem 2.3.12, we have that

$$s_i^q = (\sum_{j=1}^{\mu} e_{l_j} z_j^i)^q = \sum_{j=1}^{\mu} e_{l_j}^q z_j^{iq} = \sum_{j=1}^{\mu} e_{l_j} z_j^{iq} = s_{iq}. \tag{3.8}$$

The system of equations (3.7) is nonlinear in the z_j with unknown coefficients e_{l_j}. The strategy is to use (3.7) to set up a linear system involving new variables in order to find the error locations z_j. Once these are known, we return to the system (3.7), which is then a linear system in the e_{l_j}, and solve it to get the errors values.

Let $\sigma_0 = 1$ and let $\sigma_1, \sigma_2, \ldots, \sigma_\mu$ be the elementary symmetric polynomials in the variables $z_1, \ldots, z_\mu$ over $\mathbb{F}_q$. Then

$$\prod_{j=1}^{\mu}(x - z_j) = \sum_{j=0}^{\mu}(-1)^j \sigma_{\mu-j} x^j.$$

Moreover, by Theorem 2.3.27, we have that

$$(-1)^\mu \sigma_\mu s_j + (-1)^{\mu-1}\sigma_{\mu-1} s_{j+1} + \cdots + (-1)\sigma_1 s_{j+\mu+1} + s_{j+\mu} = 0, \quad \text{for } j = 1, 2, \ldots, \mu.$$

Proposition 3.3.21. *The system of equations*

$$(-1)^\mu \sigma_\mu s_j + (-1)^{\mu-1}\sigma_{\mu-1} s_{j+1} + \cdots + (-1)\sigma_1 s_{j+\mu+1} + s_{j+\mu} = 0, \quad j = 1, 2, \ldots, \mu$$

in the unknowns $(-1)^i\sigma_i$, $i = 1, 2, \ldots, \mu$, is solvable uniquely if and only if μ errors occur.

Proposition 3.3.22. *The system of equations*

$$\sum_{j=1}^{\mu} e_{l_j} z_j^i = s_i, \quad i = 1, 2, \ldots, \mu$$

in the unknowns e_{l_j} is solvable uniquely if the z_j are all distinct elements of $\mathbb{F}_{q^m}$.

Definition 3.3.23. The polynomial $L_{\mathbf{e}}(x)$ having as zeros the reciprocal of the error locations is said the *classical error locator polynomial* associated to the error vector $\mathbf{e}$, i.e.

$$L_{\mathbf{e}}(z) = \prod_{j=1}^{\mu}(1 - z_j x) = \sum_{j=0}^{\mu} \sigma_j x^j,$$

whereas we call the polynomial having as zeros the error locations the *plain error locator polynomial* associate to the error vector $\mathbf{e}$, and we denote it with $\mathfrak{L}_{\mathbf{e}}(z)$. Then

$$\mathfrak{L}_{\mathbf{e}}(z) = \prod_{j=1}^{\mu}(x - z_j).$$

It is clear that once we found $\mathfrak{L}_{\mathbf{e}}(z)$ (or $L_{\mathbf{e}}(z)$) then the error locations z_j may be found by computing $\mathfrak{L}_{\mathbf{e}}(\alpha^h)$ for $h = 0, \ldots, n-1$, since

$$\mathfrak{L}_e(\alpha^h) = 0 \longleftrightarrow \alpha^h \text{ is an error location.}$$

We summarize the approach to decode BCH codes that we have just shown.

Algorithm 1 Decoding BCH codes

Step 1. Compute the syndromes of the received vector $\mathbf{v}$

$$s_i = \sum_{j=1}^{\mu} e_{l_j}(z_j)^i, \quad \text{for } 1 \leq i \leq 2t$$

Step 2. Determine the maximum number $\mu \leq t$ such that the system of equations

$$s_{j+\mu} + s_{j+\mu-1}\tau_1 + \cdots s_j\tau_\mu = 0, 1 \leq j \leq \mu,$$

in the variables $\tau_i = (-1)^i\sigma_i$, has nonsingular coefficient matrix, thus obtaining the number μ of errors that have occurred. Then set up the classical error locator polynomial

$$L_e(z) = \prod_{j=1}^{\mu}(1 - z_j x) = \sum_{j=0}^{\mu} \tau_j x^j.$$

Find the coefficient τ_j from the s_i.

Step 3. Solve $L_e(x) = 0$ by substituting the powers of α into $L_e(x)$, thus finding the error locations z_j.

Step 4. Substitute the z_j in the first μ equations of Step 1 to obtain the error values e_{l_j}. Finally find the transmitted codeword $c(x) = v(x) - e(x)$.

Efficient implementations of this decoding algorithm combine the following methods

- the Horner's rule for Step 1, which requires $2tn$ multiplications in $\mathbb{F}_{q^m}$;
- the Berlekamp-Massey algorithm [Bla03, HV95] for Step 2, which costs $O(t^2)$;
- the Chien search [Chi64] for Step 3, requiring $O(tn)$ multiplications in $\mathbb{F}_{q^m}$;
- the Forney's algorithm [For65, HV95] for Step 4, costing $O(t^2)$.

Recently, further improvements [SER11] have been achieved for performing Step 1 and Step 3. If $q = p^s$ with p prime, then Step 1 can be accomplished with approximately $2s\sqrt{n(p-1)}$ multiplications and Step 3 in $\max(O(t\sqrt{n}), O(t\log(\log(t))\log(n)))$. Note that we can bound the total cost of Algorithm 1 with $O(n^2)$.

Decoding Cyclic Code up to the actual minimum distance

Cyclic codes are not known to be asymptotically bad as the BCH codes. However, up to date, there are no general efficient algebraic procedures for correcting the whole class of cyclic codes up to the actual minimum distance, i.e. up to $t = \lfloor \frac{(d-1)}{2} \rfloor$ errors, where d is the distance of the cyclic code.
Note that the previous algebraic decoding procedure designed for BCH codes can be used to decode all types of cyclic codes, but, in the general case, it can only correct up to $t_{BCH} = \lfloor \frac{(d_{BCH}-1)}{2} \rfloor$ errors, where d_{BCH} is the BCH bound of the given code. The problem in correcting t errors using the previous decoding strategy is that all the know methods solving Step 2 require knowing $2t$ consecutive syndromes. Unfortunately, for an arbitrary cyclic code the number of consecutive known syndromes is less than $2t$. Several suggestions were made in the last fifty years to resolve this problem. In the following we summarize the main approaches.

When few unknown syndromes are needed to get $2t$ consecutive syndromes, a good strategy could be to develop an efficient method determining expressions of unknown syndromes in terms of known syndromes. In [FT94] Feng and Tzeng proposed a matrix method for finding unknown syndromes representations, which is based on the existence of a syndrome matrix with a particular structure. This method depends on the weight of the error pattern, so it leads to a step-by-step decoding algorithm, and hence the error locator polynomial may not be determined in one step. In [HRTC01] He et al. developed a modified version of the Feng-Tzeng method, and used it to determine the needed unknown syndrome and to decode the binary quadratic residue (QR) code of length 47. In [CTR$^+$03, TSS$^+$08, TCCL05] Chang et al. presented algebraic decoders for other binary QR codes combining the Feng-He matrix method and the BM algorithm. Another method used to yield representations of unknown syndromes in terms of known syndromes is the Lagrange interpolation formula (LIF)[CL10]. This method has two main problems: it can be applied only to codes generated by irreducible polynomials and its computational time grows substantially as the number of errors increases. The first problem was overcome by Chang et al. in [CLF12]. Here the authors introduced a multivariate interpolation formula (MVIF) over finite fields and used it to get an unknown syndrome representation method similar to that in [CL10]. Later, trying to overcome the second problem, Lee et al.[LCJM12] presented an algorithm which combines the syndrome matrix search and a modified Chinese remainder theorem. Compared to the Lagrange interpolation method, this substantially reduces the computational time for binary cyclic codes generated by irreducible polynomials.

Besides the unknown syndrome representation method, other approaches have been proposed to decode cyclic codes. In 1987 Elia [Eli87] proposed a seminal efficient algebraic decoding algorithm for the Golay code of length 23. In 1990 Cooper [Coo90, Coo91] suggested to use Gröbner basis computations in order to deduce error locator polynomials of binary BCH codes. Cooper's idea consisted in interpreting the error locations of the code as the roots of the syndrome equations (3.7) and, in showing that the plain error locator polynomial corresponds

to the univariate polynomial $g_u(Z)$ in the normalized reduced Gröbner basis (with respect to the lexicographic term ordering induced by $z_1 < \cdots < z_t$) of the ideal generated by the polynomials in the variables $z_1, \ldots, z_t$,

$$f_i := \sum_{j=1}^{t} z_j^{2i-1} - s_{2i-1}, \quad 1 \leq i \leq t. \tag{3.9}$$

Later, the Cooper's approach [MO09] has been studied for all cyclic codes and refined [CM02, CRHT94c, CRHT94a, CRHT94b, LY97] up to give both online decoders [ABF07, ABF09] and general error locators [OS05, OS07]. Below, we briefly describe these two approaches.

Online decoders In [ABF09], Augot et al. proposed two online Gröbner basis decoding algorithms, based on Newton's identities and Waring formulas respectively, requiring, for each received vector, up to t Gröbner basis computations: the μth computation deduce the unknown $\sigma_1, \ldots, \sigma_\mu$ in terms of the known syndromes $s_1, \ldots, s_{n-k}$ of the received vector.

General locators In [CRHT94b], associating variables $Z = (z_1, \ldots, z_t)$ to the error locations, $X = (x_1, \ldots, x_{n-k})$ to the known syndromes s_i, and $Y = (y_1, \ldots, y_t)$ to the error values, Chen et al. introduced the ideal generated by the polynomials f_i and the polynomials

$$\lambda_h := y_h^{q-1} - 1, \quad \eta_h := z_h^{n+1} - z_h, \quad \text{for } 1 \leq h \leq t. \tag{3.10}$$

Note that to accommodate for the case when the number of errors, μ, is strictly less than t, it is often convenient to add *ghost error locations*, that is, $t - \mu$ zero values for the z_h's. This does not affect the value of the syndromes. The polynomials λ_h, for $1 \leq h \leq t$, specify that the error values belong to $\mathbb{F}_q^*$, whereas the polynomial η_h, for $1 \leq h \leq t$, that the locations α^{l_j}'s are nth root of unity or zero. The algorithm proposed by Chen et al. consists in deducing via a Gröbner basis computation in $\mathbb{F}_q[X, Y, Z]$, some polynomials $g_\mu(X, z)$, for $\mu \leq t$, such that, for any error with weight μ and associate known syndromes $s_1, \ldots, s_{n-k} \in \mathbb{F}_{q^m}$, g_μ is the plain error locator polynomial. In [CM02] Caboara and Mora improve this algorithm. In particular, they added to the system of equations (3.9) and (3.10), the equations

$$\theta_j := x_j^{q^m} - x_j, \quad \text{for } 1 \leq j \leq n - k. \tag{3.11}$$

The syndromes satisfy (3.11) since we have that $s_j \in \mathbb{F}_{q^m}$, for $1 \leq j \leq n - k$. The ideal generated by this new system of equations (3.9), (3.10) and (3.11) is called the *CRHT-syndrome ideal* associated to the code. The role of the polynomials λ_h, η_h, for $1 \leq h \leq t$ and $\theta_j \in \mathbb{F}_{q^m}$, for $1 \leq j \leq n - k$, is essentially to remove among the roots of the system, those roots which live in algebraic extensions and to make the other roots simple.

Both the Gröbner basis-based approaches in [ABF09] and in [CM02] are not-necessarily feasible, the first one because it requires an on line Gröbner basis computation, the second one because the system of equations has to many roots so that the Gröbner basis is less feasible to compute. In order to improve the second approach, Orsini and Sala [OS05] restricted the set of roots of the system to the point $(s_1, \ldots, s_{n-k}, z_1, \ldots, z_t, y_1, \ldots, y_t)$ corresponding to correctable syndrome vectors $\mathbf{s} = (s_1, \ldots, s_{n-k})$, by adding to the system, the following equations

$$z_h z_{h'} p_{h,h'} = 0, \quad \text{where} \quad p_{h,h'} = (z_h^n - z_{h'}^n)/(z_h - z_{h'}), 1 \leq h < h' \leq t, \tag{3.12}$$

which guarantee that for all h and h' with $1 \leq h < h' \leq t$, either z_h and $z_{h'}$ are distinct or both are zero. Studying the structure of syndrome ideal obtained by adding equations (3.12) to the CRHT-syndrome ideal, they furthermore proved the existence, for any cyclic code, of a computable polynomial, called a *general error locator polynomial*, whose roots give the error locations once it has been specialized to a given syndrome vector [OS05]. Thus, they exhibited a decoding algorithm for cyclic codes whose efficiency depends on the sparsity of this polynomial. In [OS07] Orsini and Sala gave some theoretical results on the shape of such polynomials without the need to actually compute a Gröbner basis. In particular they provided a sparse implicit representation of general error locator polynomials for all binary cyclic codes with length less than 63 and error correction capability less or equal than 2, and show that most of these codes may be grouped in a few classes, each allowing a theoretical interpretation for an explicit sparse representation of such locators. Moreover, for the codes which are not in any of these classes, they obtained a sparse explicit representation of general error locator polynomials using direct computer computations [OS07]. The low computational complexity of the general error locator polynomial for the two error-correctable cyclic codes has motivated the studies for variations on this polynomial [MOS12, LCCC10, Lee11].

In the next subsection we will focus our attention on general error locator polynomials for cyclic codes recalling some results from [OS05] and [OS07].

General error locator polynomials

The aim of this subsection is to introduce general error locator polynomials for cyclic codes and to provide a set of properties of this locators.

Let C be an $[n, k, d]$ cyclic code over $\mathbb{F}_q$ with parity-check matrix H defined by (3.4) where α is a primitive nth root of unity in $\mathbb{F}_{q^m}$, $m = \mathrm{ord}_n(q)$. Let $t = \lfloor (d-1)/2 \rfloor$ be the error correction capability of C. Let also R be the set of roots of unity in $\mathbb{F}_{q^m}$, that is $R = \{\beta \in \mathbb{F}_{q^m} \mid \beta^n = 1\}$.

Definition 3.3.24 ([OS05, Definition 3.1])**.** Let $\mathcal{L}_C$ be a polynomial in $\mathbb{F}_q[X, z]$, where $X = (x_1, \ldots, x_{n-k})$. Then $\mathcal{L}_C$ is a *general error locator polynomial* of C if

1) $\mathcal{L}_C(X, z) = z^t + a_{t-1}(X)z^{t-1} + \cdots + a_0(X)$, with $a_j \in \mathbb{F}_q[X]$, for $0 \leq j \leq t-1$, that

is, $\mathcal{L}_C$ is a polynomial of degree t with respect to the variable z and its coefficients are in $\mathbb{F}_q[X]$;

2) given a correctable syndrome $\mathbf{s} = (\bar{s}_1, \ldots, \bar{s}_{n-k})$, if we evaluate the X variables in $\mathbf{s}$, then the t roots of $\mathcal{L}(\mathbf{s}, z)$ are the μ error locations plus zero counted with multiplicity $t - \mu$.

Notice that Definition 3.3.24 can be extended to any linear code. However, given a generic linear code C the existence of a polynomial $\mathcal{L}_C$ is not guaranteed [OS05]. On the contrary, this is true for any cyclic code.

Theorem 3.3.25 ([OS05, Theorem 6.9])**.** *Each cyclic code C possesses a general error locator polynomial $\mathcal{L}_C$.*

By point 2) of Definition 3.3.24, for every correctable syndrome $\mathbf{s}$, we have that $\mathcal{L}_C(\mathbf{s}, z) = z^{t-\mu}\mathfrak{L}_{\mathbf{e}}(z)$, where $\mathbf{e}$ is the error associated to the syndrome $\mathbf{s}$. Then, it is clear that once we have computed $\mathcal{L}_C = z^t + a_{t-1}(X)z^{t-1} + \cdots + a_0(X)$ for a code C, the plain error locator polynomial may be obtained by

$$\mathfrak{L}_{\mathbf{e}}(z) = \frac{\mathcal{L}_C(\mathbf{s}, z)}{z^{t-\mu}}$$

Therefore we can resolve Step 2 of Algorithm 1 with the following procedure

Algorithm 2 Given a code C and a correctable syndrome $\mathbf{s} = (s_1, \ldots, s_{n-k})$, the procedure returns the plain error locator polynomial.

function $(C, s_1, \ldots, s_{n-k})$
  $\mu = t$
  while $a_{t-\mu}(s_1, \ldots, s_{n-k}) = 0$ **do** $\mu := \mu - 1$
  end while
  return $\mu, \mathfrak{L}_{\mathbf{e}}(z)$
end function

Note also that, since the polynomial $\mathcal{L}_C$ does not depend on the errors actually occurred, it can be precomputed once and for all. As a consequence, a decoding algorithm based on general error locator polynomial is made up of the following steps:

1. Compute the syndrome vector $\mathbf{s}$ corresponding to the received vector $\mathbf{r}$
2. Evaluate $\mathcal{L}_C$ at the syndrome vector $\mathbf{s}$
3. Apply Chien search on $\mathcal{L}_C(\mathbf{s}, z)$ to compute the error locations $\alpha^{l_1}, \ldots, \alpha^{l_\mu}$
4. Compute the error values $e_{l_1}, \ldots, e_{l_\mu}$

This approach is efficient as long as the evaluation of $\mathcal{L}_C$ is efficient. Since it is well-known that the cost of evaluating a polynomial is strictly related to the sparsity of the polynomial, one would like that each cyclic code possesses a sparse general error locator polynomial $\mathcal{L}_C$. At present, there is no theoretical proof of the sparsity of general error locator polynomials for arbitrary cyclic codes (and no proof for sparse representations), but there is some experimental evidence in the binary case. The proof of its sparsity in the general case would be a significant result in complexity theory, because it would imply that the complexity of the bounded-distance decoding problem for cyclic codes (allowing unbounded preprocessing) is polynomial in the code length.

In the following we recall some techniques used in [OS07] to efficiently compute general error locator polynomials for *binary* cyclic codes without using Gröbner bases. We also report some results on the shape of these locators for the same class of codes.

Theorem 3.3.26 ([OS07, Theorem 12])**.** *Let C,C' and C'' be three codes with the same length n and the same error correction capability t. Let $\mathcal{L}_C$,$\mathcal{L}_{C'}$ and $\mathcal{L}_{C''}$ denote their respective general error locator polynomials.*

If C is a subcode of C', then we can assume $\mathcal{L}_C = \mathcal{L}_{C'}$.

If C is equivalent to C'' via the coordinate permutation function $\phi : \mathbb{F}_2^n \to \mathbb{F}_2^n$, then we can decode C using $\mathcal{L}_{C''}$ (via ϕ).

Let C be a binary cyclic code with error correction capability $t = 2$. Let $\mathbf{s}$ be a correctable syndrome, and $\bar{z}_1$ and $\bar{z}_2$ be the (possibly ghost) error locations corresponding to the syndrome $\mathbf{s}$. Then, by Definition 3.3.24, we have that

$$\mathcal{L}_C(X, z) = z^2 + az + b, \quad a, b \in \mathbb{F}_2[X],$$

with

$$b(\mathbf{s}) = \bar{z}_1\bar{z}_2, \quad a(\mathbf{s}) = \bar{z}_1 + \bar{z}_2.$$

Moreover, there are exactly two errors if and only if $b(\mathbf{s}) \neq 0$, and there is exactly one error if and only if $b(\mathbf{s}) = 0$ and $a(\mathbf{s}) \neq 0$ (in this case the error location is $a(\mathbf{s})$). Note that if $1 \in \tilde{S}_C$ then $a(\mathbf{s}) = \bar{z}_1 + \bar{z}_2 = s_1$. So, one has that

$$1 \in \tilde{S}_C \longrightarrow \mathcal{L}_C(X, z) = z^2 + x_1 z + b, \quad b \in \mathbb{F}_2[X] \ \text{ s.t } b(\mathbf{s}) = \bar{z}_1\bar{z}_2$$

For $0 \leq \mu \leq t$, we denote by $\mathcal{V}^\mu$ the set of syndromes corresponding to μ errors. We denote also by $\mathcal{V}$ the set of *correctable syndromes*. We have that $\mathcal{V}$ is given by the (disjoint) union of the sets $\mathcal{V}^\mu$ for $\mu = 0, \ldots, t$, i.e.

$$\mathcal{V} = \mathcal{V}^0 \sqcup \mathcal{V}^1 \sqcup \cdots \sqcup \mathcal{V}^t.$$

Definition 3.3.27. Let C be a binary cyclic code code with $t = 2$. A polynomial $\mathsf{h}(X)$ in $\mathbb{F}_2[x_1, \ldots, x_{n-k}] = \mathbb{F}_2[X]$ is called a *bordering polynomial* for C if

$$\mathsf{h}(\mathcal{V}^1) = \{0\}, \quad \mathsf{h}(\mathcal{V}^2) = \{1\}\,.$$

Remark 3.3.28. Notice that if S_C contains 0 then a bordering polynomial for C is given by $(1 + x_0)$ with x_0 the variable corresponding to the syndrome $s_0 = z_1^0 + z_2^0$.

The importance of bordering polynomials comes from the following result, which immediately follows from Proposition 19 in [OS07].

Proposition 3.3.29. *Let C be a binary cyclic code with $t = 2$. Let $\mathcal{L}^* = z^2 + a(X)z + b^*(X)$ be a polynomial in $\mathbb{F}_2[x_1, \ldots, x_{n-k}, z] = \mathbb{F}_2[X, z]$, s.t. $\mathcal{L}^*(\mathbf{s}, z)$ is an error locator polynomial for any weight-2 error corresponding to a syndrome $\mathbf{s} \in \mathcal{V}^2$. Let $\mathcal{L}'_C = z + a(X)$ be an error locator polynomial for any weight-1 error, and h be a bordering polynomial. If either h or b^* is zero at the zero syndrome vector, then*

$$\mathcal{L}_C(X, z) \;=\; z^2 + a(X)z + b^*(X)\mathsf{h}(X)$$

is a general error locator polynomial for C.

In [OS07] a complete classification of all binary cyclic code with $t \leq 2$ and $n \leq 63$, with respect to their defining sets, is given.

Theorem 3.3.30 ([OS07, Theorem 28])**.** *Let C be an $[n, k, d]$ binary cyclic code with $d \in \{5, 6\}$ and $7 \leq n < 63$ (n odd). Then there are seven cases:*

1. *either n is such that the code with defining set $\{0, 1\}$ has distance at least 5,*
2. *or C is a BCH code, i.e. $S_C = \{1, 3\}$,*
3. *or C admits a defining set of type $S_C = \{1, n - 1, l\}$, with $l = 0, n/3$,*
4. *or C admits a defining set of type $S_C = \{1, n/l\}$, for some $l \geq 3$,*
5. *or C is one of the following exceptional cases*
   a) $n = 31$, $S_C = \{1, 15\}$,
   b) $n = 31$, $S_C = \{1, 5\}$,
   c) $n = 45$, $S_C = \{1, 21\}$,
   d) $n = 51$, $S_C = \{1, 9\}$,
   e) $n = 51$, $S_C = \{0, 1, 5\}$.
6. *or C is a sub-code of one of the codes of the above cases,*
7. *or C is equivalent to one of the codes of the above cases.*

According to this classification, the authors in [OS07] provide an explicit general error locator polynomial for all the codes in the first five cases. For the codes in cases 6. and 7., such locators can be gotten from those of the previous cases, as described in Theorem 3.3.26.

Note that for any binary cyclic code C (up to equivalence) in the range covered by Theorem 3.3.30, i.e. with $t = 2$ and $7 \leq n < 63$, C has a general error locator polynomial of type

$$\mathcal{L}_C(X) = z^2 + x_1 z + b(X),$$

where x_1 is the syndrome corresponding to 1, since $1 \in S_C$ in all the cases. So, for all these codes, finding $\mathcal{L}_C$ corresponds to find the polynomial $b(X)$.
For all the codes in the first four cases of Theorem 3.3.30, the shape of $b(X)$ has been theoretically determined and proved to be sparse [OS07]. The codes in 5. of Theorem 3.3.30 are those for which the general error locator polynomials are not theoretically justified within [OS07], and are solely obtained with a Gröbner basis calculation.
To theoretically justified all such error locator polynomials and to generalize this classification for binary cyclic codes with higher correction capability would be an interesting open problem.

Another useful definition from [OS07] is that of $s2ec$ code.

Definition 3.3.31. Let C be a binary cyclic code. We say that C is a *strictly-two-error-correcting code* (briefly $s2ec$ code) if, when we know that exactly two errors have occurred (that is $\mu = 2$), then we can correct them.

Trivially every binary cyclic code with distance $d \geq 5$ is a $s2ec$ code, however there are $s2ec$ codes with distance $d = 3$, e.g. the binary cyclic code defined by $n = 9$ and $S_C = \{1\}$. Note that if C is a $s2ec$ code, then every subcode C' of C is a $s2ec$ code.

Lemma 3.3.32 ([OS07, Lemma 43])**.** *Let C be a binary cyclic code with distance d and defining set $S_C = \{0, 1\}$. Then, the following are equivalent:*

1. *C is a $s2ec$ code*
2. *$d \geq 5$.*

The following lemma characterizes $s2ec$ codes.

Lemma 3.3.33 ([OS07, Lemma 45])**.** *Let C be a binary cyclic code with $S_C = \{1\}$ and length n. The following are equivalent:*

1. *for any $\{\tilde{z}_1, \tilde{z}_2\}$ and $\{\bar{z}_1, \bar{z}_2\}$ subsets of R such that $\tilde{z}_1 + \tilde{z}_2 = \bar{z}_1 + \bar{z}_2$, we have $\{\tilde{z}_1, \tilde{z}_2\} = \{\bar{z}_1, \bar{z}_2\}$*
2. *C is a $s2ec$ code*

In [OS07] the authors provide the following theorem on the structure of the general error locator polynomials of a class of binary cyclic codes.

Theorem 3.3.34 ([OS07, Theorem 52])**.** *Let C be a binary cyclic code with length n, with n $s2ec$, and let $\mathcal{L}^*$ be the error locator polynomial able to correct two errors. Suppose that $1 \in \tilde{S}_C$. Then $\mathcal{L}^* = z^2 + x_1 z + b^*(x_1)$. Moreover there exists a polynomial $A \in \mathbb{F}_{q^m}[y]$ such that*

$$b^*(x_1) = A(x_1^n)x_1^2.$$

A generalization of this result is given in [CL10]. Let $T_t = \{(\alpha^{i_1}, \ldots, \alpha^{i_j}, 0, \ldots, 0) \mid 0 \leq i_1 < \cdots < i_j < n,\ j \leq t\}$.

Theorem 3.3.35 ([CL10, Theorem 1])**.** *Let C be a binary cyclic code with length n and error correction capability t. Let also $f \in \mathbb{F}_2[x_1, \ldots, x_t]$ be a symmetric homogeneous function of degree r. Then there is a polynomial $A \in \mathbb{F}_2[x]$ such that*

$$f(z_1, \ldots, z_t) = A((z_1 + \cdots + z_t)^n)(z_1 + \cdots + z_t)^r, \quad \textit{for all } (z_1, \ldots, z_t) \in T_t.$$

Correlation Attacks on LFSR-based Stream Ciphers

A way to provide secrecy and authenticity in electronic communication data is by use of cryptographic systems. There are two basic such systems: symmetric and asymmetric cryptosystems. At the same time, symmetric encryption consists of two families of schemes, which are block ciphers and stream ciphers. This last classification may be seen in analogy to error correcting codes which are classified into block and tree codes. Block ciphers divide the plaintext message into blocks, usually of fixed size, and operate with a transformation on each block independently. Stream ciphers, in contrast, operate with a time-varying transformation on individual plaintext digits. Stream ciphers are usually faster and have a lower hardware complexity than block ciphers. This make them suitable for encrypting data over communications channel such as mobile phone and the Internet, where buffering is limited.

The purpose of this chapter is to give an outline of the background information regarding correlation attacks on certain stream ciphers. In the first section we recall some basic notions in Cryptography, including the main scenarios of attacks on cryptographic systems. In the same section we also review current algorithms for solving the Generalized Birthday Problem along with their complexities. In Section 4.2 we describe some basic properties of LFSR-based stream ciphers, their security level and the standard procedures for achieving a good level of security. In the last section we give an introduction to correlation attacks and fast correlation attacks on LFSR-based stream ciphers.

4.1 Preliminaries

4.1.1 Basic notions in Cryptography

A cryptographic scheme is a technique for transforming data from one form to another in a such way that only authorized people can recover, using a private key, the original data from the output of the transformation. Formally, we can describe such a system using the following definition.

Definition 4.1.1. A cryptosystem is comprised of three algorithms $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ such that:

- Gen is the *key-generation algorithm* that outputs the keys in the key space $\mathcal{K}$.
- Enc is the *encryption algorithm* that takes as input a message $m \in \mathcal{M}$, where $\mathcal{M}$ is the message space, and a key $k \in \mathcal{K}$, and outputs a ciphertext c in the ciphertext space $\mathcal{C}$. We

denote by $c = E_k(m)$ the encryption of the plaintext m using the key k.

- Dec is the *decryption algorithm* that takes a key $k' \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and outputs a plaintext m. We denote by $D_{k'}(c)$ the decryption of the ciphertext c using the key k'.

The basic correctness requirement of any encryption scheme is that for each $k \in \mathcal{K}$ there exists $k' \in \mathcal{K}$, and for any message $m \in \mathcal{M}$, it holds that:

$$D_{k'}(E_k(m)) = m. \tag{4.1}$$

A cryptosystem is called *symmetric* if for each $k \in \mathcal{K}$, the above key k' corresponds to k or it can be obtained from k with a simple transformation.

Given some encrypted data, we call a malicious entity whose goal is recovering the plaintext or the key, an *attacker*. There are two general approaches to attack a cryptosystem: brute-force attack (or exhaustive key search) and non-brute-force attack. In brute-force attack the attacker tries decrypting the known ciphertext c with each possible key in turn, until the correct key k is found. Note that exhaustive key search assumes that the attacker can tell whether a given guess of the key is correct. This requires that they have some information about the plaintext.
The level of security offered by a cipher is measured by the knowledge and facilities an attacker requires to attack the system using a non-brute force attack. A common assumption (Kerkhoff's principle) is to suppose that the attacker has complete knowledge of the structure of the cryptosystem. The main models of attacks from the least powerful to the most powerful (or equivalently, from the most practical to the most hypothetical) are the following:

1. *Ciphertext-only attack*: if the attacker knows one or more ciphertexts under the same key
2. *Known-plaintext attack*: in this case the attacker knows one or more matching pairs of plaintext-ciphertext
3. *Chosen-plaintext attack*: when the attacker is able to choose plaintexts and to obtain the corresponding ciphertexts

4.1.2 Boolean functions

Many stream ciphers use one or more Boolean functions as components. We recall here some definitions about Boolean functions useful for subsequent discussions.

Definition 4.1.2. A *Boolean function* f in n variables is a map from $\mathbb{F}_2^n$ to $\mathbb{F}_2$.

A Boolean function in n variables can be expressed as a polynomial in

$$\mathbb{F}_2[x_1, \ldots, x_n]/(x_1^2 - x_1, \ldots, x_n^2 - x_n),$$

called the *algebraic normal form* (ANF) of f, that is

$$f(x) = \sum_{a \in \mathbb{F}_2^n} c_a x_1^{a_1} \cdots x_n^{a_n},$$

where $c_a \in \mathbb{F}_2$ and $a = (a_1, \ldots, a_n)$.

Next definition introduces a rough measure of the complexity of a Boolean function.

Definition 4.1.3. Let f be a Boolean function. The degree of the ANF of f is called the *algebraic order* of f.

All Boolean functions of algebraic order greater than one are known as *nonlinear functions*.

Definition 4.1.4. An *affine function* $\ell_{a,c}$ in n variables is a Boolean function that takes the form

$$\ell_{a,c}(x) = a \cdot x \oplus c = a_1 x_1 \oplus \cdots \oplus a_n x_n \oplus c,$$

where $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$, $c \in \mathbb{F}_2$. If $c = 0$, then $\ell_{a,0}$ is said a *linear* function.

Another well-known representation of a Boolean function is its true table.

Definition 4.1.5. Let f be a Boolean function in n variables. We call the *truth table* of f the vector $y_f \in \mathbb{F}_2^{2^n}$ defined by

$$y_f := (f(v_0), f(v_1), \ldots, f(v_{2^n-1}))$$

where $v_0 = (0, \ldots, 0, 0), v_1 = (0, \ldots, 0, 1), \ldots, v_{2^n-1} = (1, \ldots, 1, 1)$, ordered by lexicographical order.

Definition 4.1.6. Let f be a Boolean function. The *Hamming weight* of f is defined to be the Hamming weight of its truth table, and it is denoted by $wt(f)$. Moreover, f is said *balanced* if its Hamming weight is 2^{n-1}.

Definition 4.1.7. The *Hamming distance* between two Boolean functions $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$, denoted by $\mathrm{d}(f, g)$, is defined as

$$\mathrm{d}(f, g) := wt(f \oplus g).$$

The *nonlinearity* of a Boolean function f , denoted by $\mathcal{N}_f$, is defined as

$$\mathcal{N}_f := \min_{\phi \in \mathcal{A}_n} \mathrm{d}(f, \phi),$$

where $\mathcal{A}_n$ is the class of all affine functions in n variables.

4.1.3 Birthday Problem

The *Birthday problem*, or *birthday paradox*, is a standard problem in probability theory which was initiated by von Mises in 1932. It is a very used paradigm in cryptography: it is encountered in symmetric and asymmetric cryptography, in cryptanalysis and provable security. Its original version concerns the probability that at least two people in a group of n people will have the same birthday. It is a paradox in the sense that it does not fit the common sense answer. Assuming that a year is always 365 days long and that each day of the year is equally probable for a birthday, the probability reaches 100% when the number of people reaches 366. However, 99.9% probability is reached with just 70 people, and 50% probability with 23 people.

A general formulation of the birthday problem is the following

Problem 1 BIRTHDAY PROBLEM
Input: Two lists L_1, L_2 of elements drawn uniformly and independently at random from $\{0,1\}^n$
Output: $x_1 \in L_1$ and $x_2 \in L_2$ such that $x_1 \oplus x_2 = 0$

When $|L_1| \times |L_2| \gg 2^n$ holds, a solution x_1, x_2 for this problem exists with good probability. In this context, we say that x_1 and x_2 collides or simply that we have a *collision*. Moreover, if the list sizes $|L_1|$, $|L_2|$ are favorably chosen, the complexity of the optimal algorithm for solving the birthday problem is $O(2^{n/2})$. To see that, let us define a join operation $\bowtie$ on lists so that $S \bowtie T$ represents the list of elements common to both lists S and T. Since $x_1 \oplus x_2 = 0$ if and only if $x_1 = x_2$, a solution of the birthday problem may be found by simply computing the join $L_1 \bowtie L_2$ as it is shown schematically in Figure 4.1.

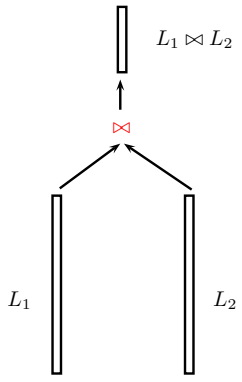

Figure 4.1: Standard algorithm for birthday problem

Given two lists L_1 and L_2, several efficient methods for computing $L_1 \bowtie L_2$ are known. In the following we mention two of these methods. A *merge-join* sorts the two lists, and scans them returning any matching pair detected. It runs in $O(M \log M)$ time where $M = \max(|L_1|, |L_2|)$. A *hash-join* stores one list in a hash table, and then scans through the elements of the other list checking whether they are present in the hash table. It has running time $O(|L_1| + |L_2|)$ and

memory complexity $O(\min(|L_1|, |L_2|))$.
Hash-join is very efficient in case of having enough memory available to store $\min(|L_1|, |L_2|)$ elements. Otherwise the merge-join is preferable since external sorting methods allow to compute it even when memory is scarce.
Then, if we are free to choose the size of the lists as we desire, birthday problem can be solved with $O(2^{n/2})$ time and space.

In [Wag02] Wagner introduced a generalization of Problem 4.1.3 for an arbitrary number of lists. He refers to the problem where there are k lists as the *k-sum problem*.

Problem 2 k-SUM PROBLEM
Input: k lists $L_1, L_2, \ldots, L_k$ of elements drawn uniformly and independently at random from $\{0,1\}^n$
Output: $x_1 \in L_1, x_2 \in L_2, \ldots, x_k \in L_k$ such that $x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0$

As the birthday problem, it is easy to see that a solution $x_1, \ldots, x_k$ for the k-sum problem exists with good probability once $|L_1| \times |L_2| \times \cdots \times |L_k| \gg 2^n$.

In 1981 Schroepple and Shamir [SS81] considered the problem to find all solutions of the k-sum problem in the case when $k \leq 4$. Their algorithm combines dynamic programming techniques with a divide-and-conquer algorithm. In particular, when $k = 4$, they decrease the time complexity of the algorithm from brute force $O(2^n)$ to $O(2^{n/4})$ space and $O(2^{n/2})$ time.

In [Wag02], Wagner develops an algorithm for solving the k-sum problem assuming that one can extend the sizes of the lists freely, i.e. in the special case where there are sufficiently many solutions to the k-sum problem. Choosing as list sizes $O(2^{n/(1+\log k)})$, Wagner's algorithm requires $O(k \cdot 2^{n((1+\log k)})$ time and space. In the following, we sketch the idea behind this algorithm for the case when $k = 4$.

Let $L_1, \ldots, L_4$ be four lists of elements chosen uniformly and independently at random from $\{0,1\}^n$. The idea is to generalize the arguments showed before for the birthday problem. Let us introduce a generalized join operator $\bowtie_l$, with $l \leq n$, so that $L_1 \bowtie L_2$ contains all pairs from $L_1 \times L_2$ that agree in their l least significant bits. Denoting by $\mathrm{low}_l(x)$ the least significant bits of x, we have $\mathrm{low}_l(x_i \oplus x_j) = 0$ if and only if $\mathrm{low}_l(x_i) = \mathrm{low}_l(x_j)$ for x_i and x_j lists of bits of the same length.
To get a solution for the 4-sum problem, firstly, we extend each list L_i until it contains about 2^l elements with $l = n/3$. Then, we generate the list L_{12} of values $x_1 \oplus x_2$ with $x_1 \in L_1$ and $x_2 \in L_2$ such that $\mathrm{low}_l(x_1 \oplus x_2) = 0$. Similarly, we generate a list L_{34} from the lists L_3 and L_4. Lists L_{12} and L_{34} can be constructed by simply computing the joins $L_1 \bowtie_l L_2$ and $L_3 \bowtie_l L_4$. Now, note that if $x_1 \oplus x_2 = x_3 \oplus x_4$ then $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. So, finally, we search for collisions between L_{12} and L_{34}. Any such collision will yield a solution to the 4-sum problem.

The resulting algorithm can be implemented with $O(2^{n/3})$ time and space. The estimation of the algorithm is based on the facts that $\Pr[\mathrm{low}_l(x_1 \oplus x_2) = 0] = 1/2^l$ when x_1, x_2 are taken uniformly at random, and that if $\mathrm{low}_l(x_1 \oplus x_2 \oplus x_3 \oplus x_4) = 0$ then $\Pr[x_1 \oplus x_2 \oplus x_3 \oplus x_4 =$

$0] = 1/2^{n-l}$. In Figure 4.2, we report schematically the algorithm for the case $k = 4$.

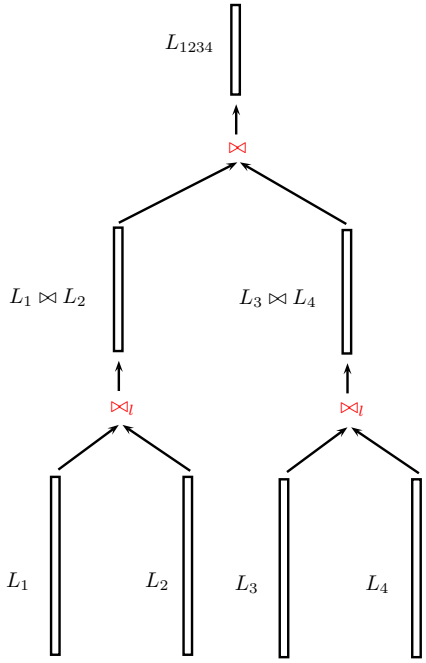

Figure 4.2: Representation of Wagner's algorithm for the 4-sum problem

For a general $k > 2$, the algorithm of Wagner can be represented as a complete binary tree of depth $\lfloor \log k \rfloor$ where the edges are lists $L_{i\ldots j}$ obtained from the lists $L_1, \ldots, L_k$ using specific join operators. At height h it is used the join operator $\bowtie_{l_h}$ with $l_h = hn/(1 + \lfloor \log k \rfloor)$ for $h = 1, \ldots, \lfloor \log k \rfloor - 1$. At the root it is used the full join operator $\bowtie$. For this reason the algorithm is also called *k-tree algorithm*.

In [CJM02], Chose, Joux and Mitton produce a space-efficient algorithm to find *all* solutions of k-sum problem based on match-and-sort alternative algorithm. Their formulation of the problem is quite different from the Wagner's one. They do not introduce the input lists $L_1, \ldots, L_k$ and their goal in presenting the algorithm is to improve the efficiency of fast correlation attacks on certain LFSR-based stream ciphers. Their algorithm runs in $O(N^{\lceil k/2 \rceil} \log N)$ time and $O(N^{\lfloor (k+1)/4 \rfloor})$ memory, where N corresponds to the maximal list size in Wagner's formulation. In particular, for $k = 4$, their approach runs $O(2^{n/2})$ time and $O(2^{n/4})$ space if $|L_1| = \cdots = |L_4| = 2^{n/4}$. This algorithm is essentially equivalent to repeatedly running Wagner's 4-tree algorithm once for each possible predicted value of $\alpha = \mathrm{low}_l(x_1 \oplus x_2)$, taking $l = n/4$.

Note that the k-tree algorithm fails when the size m of the lists is smaller than $2^n/(\log k+1)$. In [MS12], Minder and Sinclair generalize the k-three algorithm for values of m smaller than $2^n/(\log k + 1)$. The key step in the algorithm of Minder and Sinclair is to specify an optimal strategy for choosing the lengths l_i in each round and to reduce the way of finding these lengths to an integer program. Their algorithm works for all values of m such that

$2^n/k \leq m \leq 2^n/(\lfloor \log k \rfloor + 1)$ in complexity of $O(2^{\lfloor \log k \rfloor + u^*})$, where u^* is the optimal value of u, which represents the maximal list length.

4.2 Stream ciphers

Stream ciphers are symmetric ciphers which operates with a time-varying transformation on *individual* plaintext digits. More precisely, in a stream cipher a sequence of plaintext digits, $m_0 m_1 \ldots$, is encrypted into a sequence of ciphertext digits $c_0 c_1 \ldots$ as follows: the ith ciphertext digit is produced by combining the ith plaintext digit with the ith digit of a sequence obtained by the secret key k according to some rule. The sequence obtained by the key is called *keystream*. In *synchronous stream ciphers* the keystream is generated independently of the plaintext and of the ciphertext. Otherwise, we call the stream cipher *asynchronous*.
Synchronous stream ciphers are preferable respect to asynchronous ones because they are not affected by error-propagation. The most common form of synchronous stream cipher is the *binary additive stream ciphers*, where plaintexts, chipertexts and keystreams are sequences of bits and both encryption and decryption are simply the bitwise `XOR`-operation. A major advantage of these stream ciphers is that encryption and decryption can be performed by the same device. Figure 4.3 shows the encryption and decryption for a binary additive stream cipher.

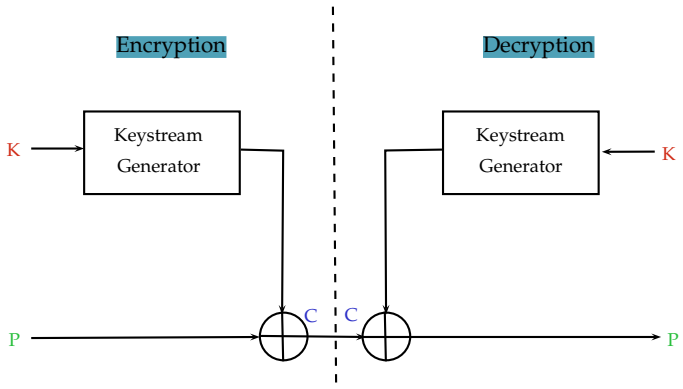

Figure 4.3: A binary additive synchronous stream cipher

In the following we trait only the case of binary additive synchronous stream ciphers. If no differently specified, when we say stream cipher we actually mean binary additive synchronous stream ciphers.

4.2.1 Standard properties of keystream sequences

One of the most remarkable of all ciphers is the *one-time-pad* or Vernam cipher. It is a stream cipher where the keystream is a non repeating random sequence of the same length as the plaintext. This cipher was developed by G. Vernam in 1917 for telegraph communications. The remarkable fact about this cipher is that it leads to perfect secrecy.

Definition 4.2.1. A cryptosystem with set of plaintexts $\mathcal{P}$ and set of ciphertexts $\mathcal{C}$, is said to have the property of *perfect secrecy* if, for all $p \in \mathcal{P}$ and $c \in \mathcal{C}$ with $\mathbb{P}(Y = c) > 0$,

$$\mathbb{P}(X = p|Y = c) = \mathbb{P}(X = p),$$

where $\mathbb{P}(Y = c)$ denotes the probability that the ciphertext is c, $\mathbb{P}(X = p)$ the probability that the plaintext is p, and $\mathbb{P}(X = p|Y = c)$ the probability that the plaintext is p given that the ciphertext is c.

The fact that the one-time-pad has the property of perfect secrecy implies that, assuming a ciphertext-only attack, even with infinite computing resources, the attacker could never distinguish the true plaintext from all the other meaningful plaintexts. The disadvantage of this cipher is that it requires that the key has the same length as the plaintext. This requirement makes it impractical in the most applications.

The appealing features of the one-time-pad suggested building synchronous stream ciphers using as keystream a deterministically generated "random" sequence. We call such a sequence a *pseudorandom sequence*. The deterministic algorithm generating the keystream is said the *keystream generator*. The security of this type of stream ciphers depends on the "randomness" of the keystream. Note that in a known-plaintext attack scenario, the attacker has full access to the keystream. So, in particular, if it is possible to predict the entire keystream from a small known segment, then the cipher offers a low level of security.

There are three standard properties of pseudorandom sequences used to measure the security of a stream cipher: the period, white noise statistics and linear complexity. In the remainder of this paragraph we briefly describe the reason why they represent the standard criterion of randomness.

A deterministic pseudorandom generator produces a sequence which is periodic (or ultimately periodic). Then, a necessary requirement for unpredictability is that the keystream has a long period. In particular the period of the keystream should be substantially larger than the length of the plaintext. Moreover one would like that the first period of the keystream looks like "random". Note that it seems difficult to define randomness for finite sequences. In [Knu81], Knuth describes several criteria to evaluate the randomness of a finite sequence of bits. One of these is the criterion of distribution properties, which we can state as follows.

Definition 4.2.2. A finite sequence of bits of length L may be called *random* if for all k smaller than some upper bound (say $< \log_2 L$) every binary k-tuple appears about equally often.

Golomb [G$^+$82] proposed three postulates based on this definition, to measure the randomness of a periodic binary sequence. But, unfortunately, the Golomb postulates do not define a general measure of randomness for finite sequences, since there are examples of finite sequence satisfying Golomb axioms which not appear random.
We recall that an efficient method for producing periodic sequences with maximum possible period is by using linear feedback shift registers (LFSRs). It turns out that maximal period

sequences have also good statistically properties. In particular they satisfy Golomb postulates. However, due to the linearity, sequences generated by LFSRs are easily predicted from a short segment: a maximal-period binary sequence of period $2^L - 1$ is completely determined by knowing $2L$ consecutive sequence bits [Mas69]. Next definition is related to this property.

Definition 4.2.3. The *linear complexity* of a binary sequence is defined to be the length of the shortest binary LFSR which can produce the sequence.

According to what we have just said, the linear complexity of the keystream should be sufficiently large. We note that the above properties are necessary requirements for suitable sequences for cryptographic applications, but in the general case they are not sufficient.

4.2.2 LFSR-based stream ciphers

The keystream generator can be viewed as a finite state machine whose initial state x_0 is derived from the secret key (and usually from a public initial value) by a key-loading algorithm. At each time unit t, the keystream digit z_t produced by the generator is the output of a Boolean function, called *filtering function*, applied to the current internal state x_t. The internal state x_t is then update by a *transition function*. Both filtering function and transition function must be chosen carefully in order to obtain a secure stream cipher.
Many stream ciphers are based on LFSRs since these devices are easily implemented in hardware and are mathematically well understood. Also, sequences produced by LFSRs with primitive characteristic polynomials, has large period and good statistical properties. However, as we noticed in the previous section, they have low linear complexity. This makes LFSRs unsuitable for use as keystream generators on their own. Since this is due to their linearity, it sufficient to incorporate some kind of nonlinearity. Nonlinearity could be introduced in keystream either explicitly or implicitly. Here we describe the four common methods.

Nonlinear combination generators

A method for constructing keystream sequences using LFSRs is combining the outputs of a certain number, n, of LFSRs by a nonlinear filtering function f. Such generators are called *nonlinear combination generators* and the filtering function f is called the *combining function* of the generator.
The cryptographic properties of the sequences produced with nonlinear combination generators are well understood [Rue86]. Suppose that the n LFSRs have primitive characteristic polynomials and that their lengths $L_1, L_2, \ldots, L_n$ are pairwise coprime. Then the period of the output sequence is $\pi = \prod_{i=1}^{n}(2^{L_i} - 1)$ and its linear complexity is $L = \bar{f}(L_1, L_2, \ldots, L_n)$, where $\bar{f}$ is the function over the integers obtained by f interpreting addition and multiplications over the integers rather than over $\mathbb{F}_2$. The period of the sequences produced by these generators can be increased by increasing the lengths of the LFSRs, while one can increase their linear complexity

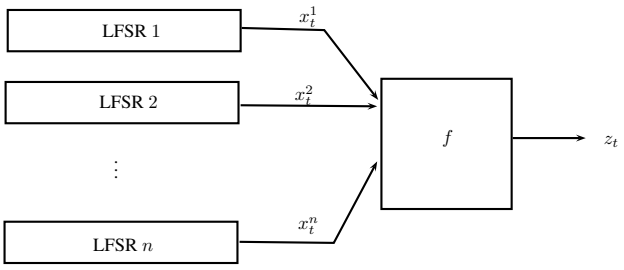

Figure 4.4: Combining generator based on LFSRs

by increasing the lengths $L_1, \ldots, L_n$ and the algebraic order of the combining function. However, for memoryless generators an increase in the algebraic order of the combining function corresponds to an increase of their vulnerability to correlation attacks.

Nonlinear filter generators

A second common method used for getting keystream sequences consists in using a nonlinear Boolean function to combine the outputs of several stages of a single regularly clocked LFSR. These generators are called *nonlinear filter generators*. If the length of the LFSR is L then the period of the output sequence is $2^L - 1$ or a divisor of $2^L - 1$, while its linear complexity is $\leq \sum_{i=1}^{k} \binom{L}{i}$, where k is the algebraic order of the filtering function. In nonlinear filter generators the filtering function f must be a balanced Boolean function.

Nonlinear combination generator with memory

A *generator with memory* can be viewed as a nonlinear combination generator with internal state (x_t, σ_t) where $x_t = (x_t^1, \ldots, x_t^n)$ is update linearly using n LFSRs and $\sigma_t = (\sigma_t^1, \ldots, \sigma_t^M)$ is update with a nonlinear function. We call the σ_t the memory of the generator. Usually M is less than n. The sequences produced by such generators are ultimately periodic with period given by $\pi = \prod_{i=1}^{n}(2^{L_i})$, where L_i, for $i = 1, \ldots, n$, is the length of the ith LFSR. Also, appropriately choosing the two transition functions, the linear complexity of the sequences is approximately equal to the length of the period, i.e. $\lceil \log_2 \pi \rceil$.

Clock-controlled generators

An other method for constructing keystream generators is by controlling the clock of the LFSRs. An LFSR-based keystream generator is said *regularly clocked* if a keystream bit is produced every time the underlying LFSRs are clocked. If the keystream bits are produced at irregular intervals then we call the generator a *clock-controlled generator*. Irregular clocking introduce nonlinearity implicitly.

We conclude this section giving a practical example of stream cipher.

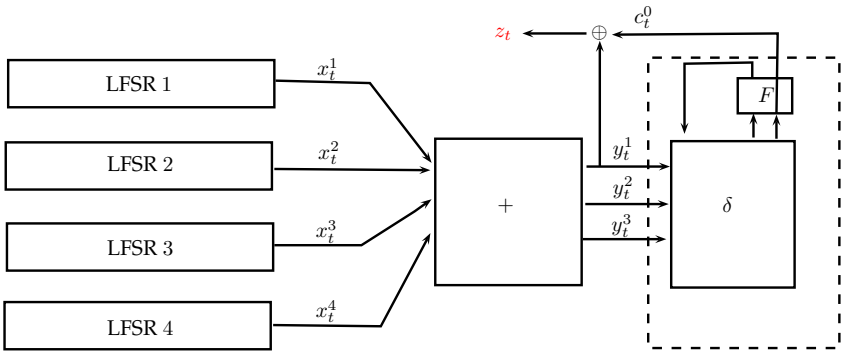

Figure 4.5: The cipher E0

The cipher E0

One of the most widely used stream ciphers is the cipher E0. It is used for securing Bluetooth communications [Blu10]. The keystream generator of E0 is a nonlinear combination generator with memory.
It consists of four LFSRs with lengths $L_1 = 25$, $L_2 = 31$, $L_3 = 33$, $L_4 = 39$ and a nonlinear combiner with 4 bits of memory, denoted by $\sigma_t = (c_{t-1}, c_t)$ at time t, where $c_t = (c_t^1, c_t^0)$. At each clock t, the four LFSRs' output bits x_t^i, for $i = 1, \ldots, 4$, are added as integers to form a sum y_t. Denote by y_t^i the ith least significant bit of the binary representation of y_t, for $i = 1, 2, 3$. A 16-state machine emits one bit c_t^0 out of its state $\sigma_t = (c_{t-1}, c_t)$ and takes the input y_t to update σ_t by σ_{t+1} using a nonlinear function. Finally, the keystream z_t is obtained by xoring y_t^1 with c_t^0, that is, by $z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0$.

4.3 Correlation attacks on LFSR-based stream ciphers

Correlation attacks were introduced by Siegenthaler in [Sie85]. They are among the most important and studied attacks on stream ciphers based on nonlinear *combination* generator. They use a divide-and-conquer strategy, breaking down the problem of key recovery into the problem of recovering subkeys. They apply when a part of the internal state of the keystream generator is update independently from the other part, and its size is reasonable. When the target part of the internal state is update linearly, a major improvement was made by Meier and Staffelbach [MS89]. This refinement led to different versions of fast correlation attacks [CT00, JJ99b, CJS01].

Here we focus on correlation attacks on LFSR-based stream ciphers, describing the main aspects of these attacks. We denote by x_t the n-bit internal state of the generator at time t, by f the filtering function and by z_t the output bit of the keystream generator at time t, i.e. $z_t = f(x_t)$. We assume a know-plaintext attack scenario which aim at recovering the initial state x_0.

4.3.1 Correlation attacks

A correlation attack against nonlinear combination generators, can be mounted when the internal state x_t of the generator is composed by two parts $x_{1,t}$ and $x_{2,t}$, having independent update functions.

Consider a combination generator based on n LFSRs, namely LFSR 1,. . ., LFSR n. Let x_t^i be the output bit of the LFSR i at time t, for $i = 1, \ldots, n$. So $x_t = (x_t^1, \ldots, x_t^n)$.
Let us fix k, with $1 \leq k < n$. A correlation attack on the k registers LFSR i_1,. . ., LFSR i_k, with $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, can be described as follows. Let $x_{1,t} = (x_t^{i_1}, \ldots, x_t^{i_k})$ and $x_{2,t} = (x_t^i)_{\substack{1 \leq i \leq n \\ i \notin \{i_1, \ldots, i_k\}}}$. We have that $x_t = (x_{1,t}, x_{2,t})$. Let Φ_1, Φ_2 be the two transition functions on $x_{1,t}$ and $x_{2,t}$ respectively. Then

$$(x_{1,t+1}, x_{2,t+1}) = (\Phi_1(x_{1,t}), \Phi_2(x_{2,t})).$$

The attack aim at recovering one of the two parts of the initial state of the generator, say $x_{1,0}$. We call this vector the *target state*. The attack require the existence of a Boolean function g of k variables which is correlated to the filtering function f, i.e. such that

$$p_g = \mathbb{P}_{X_1, X_2}[f(X_1, X_2) = g(X_1)] > 1/2, \tag{4.2}$$

where X_1 and X_2 are two independent random variables uniformly distributed in $\mathbb{F}_2^k$ and $\mathbb{F}_2^{n-k}$ respectively. A very significant fact about nonlinear functions is that such functions g always

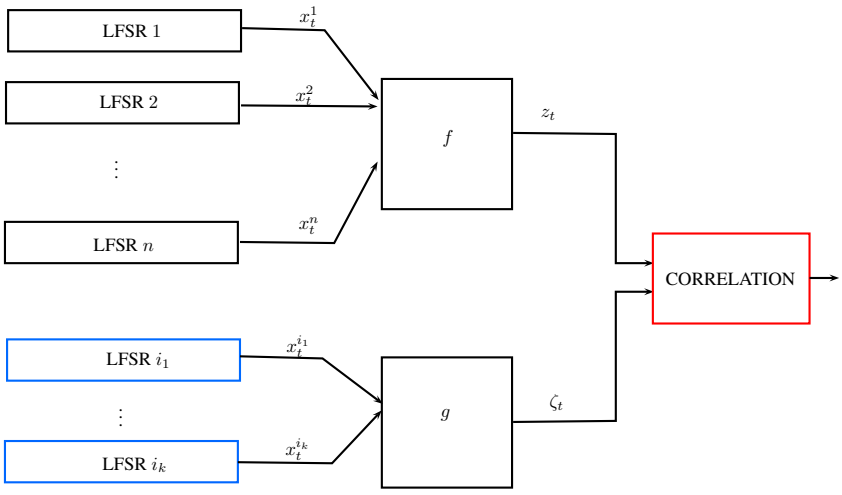

Figure 4.6: Correlation attack involving k constituent LFSRs

exist. However, for the attack to be successful, the correlation need to be good enough. The main method for finding correlations is to use statistical tests designed for distinguishing two binary random sources [CT12, Jun05]. Let g be a Boolean function satisfying (4.2). Then we say that the *target sequence* $\zeta = (\zeta_t)_{t \geq 0}$, defined by $\zeta_t = g(\Phi_1^t(\mathbf{x}_{1,0}))$, is correlated to the

keystream sequence $z = (z_t)_{t\geq 0}$. The quality of the correlation is related to the quantity p_g. The function g must be chosen such that p_g is maximal. It can be proved that

$$\max_g p_g = \frac{1}{2} + \frac{1}{2^k} \sum_{x_1 \in \mathbb{F}_2^k} |\frac{1}{2} - p_{x_1}|, \tag{4.3}$$

where $p_{x_1} = \mathbb{P}[f(x_1, X_2) = 1]$. By (4.3), it follows that the maximal value of p_g depends on the distributions of the output of f when its input variable X_1 is fixed. So correlation attacks can be prevented if the filtering function f is such that its output remains uniformly distributed also when its input variable X_1 is fixed. The maximal integer k such that a function f satisfying the previous property is called the *correlation-immunity order* of f. If the filtering function has correlation-immunity order $k-1$, then the minimum number of LFSRs which must be considered together for the attack to be successful is k. However, it is also known that there is a trade-off between the correlation-immunity order and the nonlinearity of f [Sie84]. Hence k must be rather small. It can be proved that the best target sequence ζ is obtained by adding the outputs of the k involved LFSRs, that is

$$\zeta = (\zeta_t), \quad \zeta_t = x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus x_{i_k}. \tag{4.4}$$

Note that the sequence (4.4) corresponds to the output of a unique LFSR whose characteristic polynomial is given by

$$P = \mathrm{GCD}(p_1, \ldots, p_k),$$

with p_j the characteristic polynomial of the jth LFSR, for $1 \leq j \leq k$. Moreover, assuming that the polynomials p_j, for $1 \leq j \leq k$ are primitive, the length of the LFSR producing ζ is given by

$$L = \sum_{j=1}^{k} L_j,$$

with L_j the length of the jth LFSR, for $1 \leq j \leq k$. In this circumstance, the attack requires that all $2^{\sum_{j=1}^{k} L_j}$ initial states of the target state be examined. This becomes infeasible when the correlation-immunity order $k-1$ of the combining function is high (for practical ciphers, typical values for k are $k = 1, 2$).

4.3.2 Fast correlation attacks

The fast correlation attack proposed by Meier and Staffelbach [MS88, MS89] relies on the same principle as correlation attacks but avoid performing exhaustive search for the initialization of the target state. The method proposed by Meir and Staffelbach considerably reduces the time complexity of the attack but requires the knowledge of a larger segment of the keystream.

Suppose we are in the same scenario as the previous subsection, with p_j primitive, for $1 \leq j \leq k$, and ζ defined by (4.4). Moreover, let N be the number of known keystream bits. Since ζ can be seen as the output of a LFSR with characteristic polynomial P and length

L, any subsequence of length N of ζ is a codeword of the $[N, L]$-code C having as parity-check matrix the following $(N-L) \times N$ matrix

$$H = \begin{pmatrix} c_L & c_{L-1} & \dots & c_0 & 0 & \dots & 0 \\ 0 & c_L & c_{L-1} & \dots & c_0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & c_L & c_{L-1} & \dots & c_0 \end{pmatrix}, \tag{4.5}$$

where $P(x) = c_0 + c_1 x + \cdots + c_L x^L$, with $c_0 = c_L = 1$. The key idea of fast correlation attacks consists in viewing the influence of the nonlinear combining function as a binary symmetric channel (BSC), with cross-over probability $p = \mathbb{P}[\zeta_t \neq z_t] > \frac{1}{2}$. The observed output sequence $\mathbf{z} = (z_0, z_1, \dots, z_{N-1})$ is then regarded as the received word resulting from the transmission of the unknown codeword $\zeta = (\zeta_0, \zeta_1, \dots, \zeta_{N-1})$ through this BSC. Thus, recovering the initialization of the target state consists in decoding the keystream subsequence $(z_t)_{t<N}$ relatively to the linear code C. Figure 4.7 shows the general model for a fast correlation attack. From

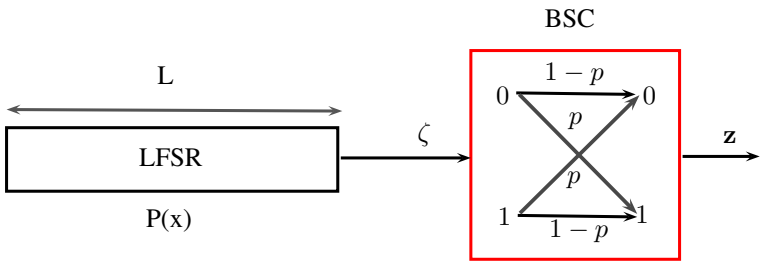

Figure 4.7: Model for fast correlation attack

the attacker's point of view, the main problem is to make the attack feasible even when a small number N of keystream bits is known. By Theorem 3.1.9, a linear code can be successfully decoded only if its rate does not exceed the capacity of the transmission channel, which for binary symmetric channel is

$$C(p) = 1 + p \log_2 p + (1-p) \log_2 (1-p).$$

Since in most practical situations p is close to $1/2$ (typical values for p are $p = 0.25, 0.3, 0.4, 0.45$), then the capacity of the channel can be approximated by $C(p) \simeq \frac{2\varepsilon^2}{\ln 2}$, where $\varepsilon = 1/2 - p$, leading to the following required keystream length:

$$N \geq \frac{L}{C(p)} \simeq \frac{\ln(2) L}{2\varepsilon^2}.$$

Unfortunately there is not known an efficient general decoding algorithm for achieve the channel capacity. This means that practical correlation attacks require the knowledge of a much longer segment of the keystream sequence than this theoretical bound. We note that any improvement of fast correlation attacks consists in finding an efficient decoding procedure for the

code C.
Note also that the original correlation attack of Siegenthaler consists in applying a maximal-likelihood decoding (ML-decoding) algorithm to the code C. For practical LFSRs lengths this decoding is infeasible. For this reason, the attacker needs to use much faster decoding algorithms.

There are two main families of fast decoding algorithms used in correlation attacks: general decoding algorithms and algorithms based on low-weight polynomial multiples. The decoding algorithms in the first family are algorithms which can be applied to any linear code. They are based on the following key-idea [CJS01]. When the code dimension L is too large for ML-decoding, it is possible to derive from the original code a new code with smaller dimension on which ML-decoding can be applied. On the contrary, the second family of algorithms exploit the particular structure of the LFSR code C defined by the parity-check matrix (4.5) using the existence of sparse *parity-check equations* for this code. This technique was first proposed by Meier and Staffelbach [MS88]. It is based on the following observation. Any sparse multiple

$$1 + x^{e_1} + \cdots + x^{e_{w-1}}$$

of the characteristic polynomial P exactly corresponds to a linear relation involving w bits of the LFSR sequence:

$$\forall t, \quad \zeta_t \oplus \zeta_{t-e_1} \oplus \cdots \oplus \zeta_{t-e_{w-1}} = 0.$$

When a collection of such sparse equations is available, the LFSR code C can be viewed as a low-density parity-check (LDPC) code. Then, some efficient iterative decoding techniques [Gal63, JJ99b, JJ99a, MFI01] can be applied to C. Note that finding sparse multiple of P corresponds to find low-weight codewords of the binary cyclic code of length $2^{\mathrm{LCM}(p_1,\ldots,p_k)} - 1$ generated by P.

Both families of algorithms involve the precomputation of many low-weight parity-check equations of the LFSR in order to speed up the decoding process. When $k = 1$ this precomputation step can be performed according to two different approaches, one based on the birthday paradox [CJM02] and another based on discrete logarithms [DLC07].

It might be argued that the design of modern stream ciphers evolved accordingly. A cipher like E0 ([GPS04]) is not immediately subject to these types of attacks, since no apparently single LFSR is correlated to the keystream output. In [LV04], Lu and Vaudenay introduced a new fast correlation attack which is able to successfully recover the state of E0. Their attack requires a different precomputation step which computes a *single* parity check of *multiple* LFSRs. The complexity of their precomputation step is not far from the complexity of their full attack, and they employed the generalized birthday approach presented in [Wag02]. Since the data complexity of their attack is bounded from below by the degree of the parity check, it is important to find a parity check of degree less than a target degree. Further research was recently presented in [PTS14], which contains in particular a straightforward generalization of the discrete logarithm approach of [DLC07].

Part II

Main results

Discrete logarithm-based approach for fast correlation attacks

Being able to compute efficiently a low-weight multiple of a given binary polynomial is often a key ingredient of correlation attacks to LFSR-based stream ciphers. The best known general purpose algorithm is based on the generalized birthday problem. In this chapter we describe an alternative approach for finding low-weight polynomial multiples which is based on discrete logarithms. Contrary to birthday-based approach it can take advantage of the structure of the given polynomial. Moreover, in some cases it has much lower memory complexity requirements with a comparable time complexity respect to generalized based approach.

In Section 5.1, we present our strategy and give algebraic results on which the algorithm is based. The algorithm we propose is explained in Section 5.2, along with a comparison of its complexity to the generalized birthday approach and to the straightforward generalization of the discrete log approach for the case of a single primitive polynomial. Significant examples of our approach are outlined in Section 5.3, where we show that for the fast correlation attack in [LV04] our algorithm could be more convenient to use in the second precomputation step than the generalized birthday approach, and that the method we propose is substantially better than the generalized birthday approach in the case where the polynomial p can be decomposed in several irreducible factors, each one of degree less than 20.

5.1 Strategy and preliminary results

The problem we will address throughout the chapter is the following:

Problem 3 FIND A GIVEN-WEIGHT POLYNOMIAL MULTIPLE WITH TARGET DEGREE
Input: A polynomial p and two integers $w \geq 3$ and D.
Output: A multiple of p of weight w and degree at most D, if it exists.

Our general strategy for the resolution of Problem 3 consists in the following three steps:

- Firstly, we translate the problem of finding a multiple of a binary polynomial into finding an appropriate sequence of integers modulo the order of a polynomial (Propositions 2.3.8,5.1.2).

- Next, we show that the general problem is solved once we know how to solve the problem for single factors of the unique factorization into irreducible polynomials (Proposition 5.1.3). The problem for a single irreducible factor is treated depending on the primitivity of the factor, but in both cases we use discrete logarithms.

- Finally, powers of irreducible polynomials could be completely characterized, although in our algorithm we will aim for a subset of all possible multiples (Remark 5.1.9) for computational reasons.

To illustrate more in details the approach we consider an example. Here we will denote a binary polynomial $x^{d_1} + x^{d_2} + \cdots + x^{d_k}$ also by the sequence of its exponents $[d_1, \ldots, d_k]$.

Example 5.1.1. Let us consider the polynomial $p = x^{17} + x^{16} + x^{15} + x^{13} + x^{12} + x^8 + x^6 + x^5 + x^3 + x + 1$ and let

$$p = (x^5 + x^3 + x^2 + x + 1)(x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1)(x^2 + x + 1)^2$$

be its factorization into irreducible polynomials. We observe that p has three factors, two of multiplicity one and one with multiplicity two. It is easy to check that the polynomials $p_1 = x^5+x^3+x^2+x+1$ and $p_3 = x^2+x+1$ are primitive while $p_2 = x^8+x^7+x^6+x^5+x^4+x^3+1$ is not. We want to exhibit a polynomial of weight 3, $f = [d_1, d_2, d_3]$ with $d_1 < d_2 < d_3$, multiple of p.

- Let q be the product of the irreducible factors of p, $q = p_1p_2p_3$. If we find a multiple of q, $f' = [e_1, e_2, e_3]$ with $e_1 < e_2 < e_3$, then $f = f'^2 = [2e_1, 2e_2, 2e_3]$ is a multiple of p. So we have reduced our search to the case where the polynomial is a product of distinct irreducible polynomials.

- In order to find f' we note that we can always set $e_1 = 0$ because q is such that $q(0) \neq 0$. Then, we pick at random all the other exponents of the target multiple f' except the last one. Since we are searching for a multiple of weight 3, in this case we pick e_2 at random with $0 < e_2 < N$ where N is the order of q. To obtain a multiple $f' = [0, e_2, e_3]$ of q we assume to be able to solve the same problem for p_1, p_2 and p_3, getting polynomials of the form $[0, e_2, e_3^{(i)}]$ for $i = 1, \ldots, 3$, and then we try merging the results applying the Chinese Remainder Theorem. If the merging is successful a multiple of q was found, otherwise we try with another e_2.

- To get multiples of the irreducible factors let us distinguish the case p_i is primitive and the case p_i is not primitive. When p_i is primitive we set $e_3^{(i)} = \mathrm{DLog}_{\alpha_i}(1 + \alpha_i^{e_2})$ with α_i a primitive root of p_i, otherwise we set $e_3^{(i)} = \mathrm{DLog}_{\alpha_i}(1 + \alpha_i^{me_2})/m$ with α_i^m a root of p_i and α_i a primitive element of $\mathbb{F}_{2^{\deg(p_i)}}$.

Let us suppose to pick $e_2 = 5$. Then we get $e_3^{(1)} = 28$, $e_3^{(2)} = 28$, $e_3^{(3)} = 1$. So the polynomials $1 + x^5 + x^{28}$, $1 + x^5 + x^{28}$ and $1 + x^5 + x$ are, respectively, multiples of p_1 p_2 and p_3. Let e_3 be the result of applying the Chinese Remainder Theorem to 28, 28 and 1 with moduli the orders of the polynomial p_1 p_2 and p_3 respectively. It is easy to check that $e_3 = 28$. Then a multiple of p is $f = (x^{28} + x^5 + 1)^2 = x^{56} + x^{10} + 1$.

This strategy will be adapted to the search of a polynomial multiple of a certain target degree D. We will draw randomly the exponents e_i in $0 < e_i \leq D$ for $2 \leq i < w$, and a further condition for termination of the algorithm will be: $e_w \leq D$.

Let $R = \mathbb{F}_2[x]$ be the ring of binary polynomials. A binary polynomial is uniquely determined by the position of its nonzero coefficients, that is, its support. We denote a binary polynomial by its exponents:

$$[e_1, e_2, \ldots, e_k] := x^{e_1} + x^{e_2} + \cdots + x^{e_k},$$

where the e_i's are non-negative integers. If two integers in the list are repeated we can omit them from the list and obtain the same binary polynomial. If no integers are repeated then $[e_1, \ldots, e_k]$ is a polynomial of weight k. From now on when we say "polynomial" we actually mean "binary polynomial".

Let $f = [e_1, e_2, \ldots, e_k]$ be a polynomial. The remainder of the division of f by the polynomial $1 + x^N$ is obtained through reduction modulo N of its exponents:

$$[e_1, e_2, \ldots, e_k] \bmod [0, N] = [\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_k], \tag{5.1}$$

where $0 \leq \bar{e}_i < N$ and $e_i \equiv \bar{e}_i \pmod{N}$.

Proposition 5.1.2. *Let f, f', p be polynomials, $f = [e_1, \ldots, e_k]$ and $f' = [e'_1, \ldots, e'_k]$ such that for any i, $e_i \equiv e'_i \pmod{M}$, where M is a multiple of $\mathrm{ord}(p)$. Then, f is a multiple of p if and only if f' is so.*

Proof. Since $e_i \equiv e'_i \pmod{M}$, by (5.1), the remainder of the division of f and f' by $1 + x^M$ is the same. Let r be this remainder. So $f = q(1 + x^M) + r$ and $f' = q'(1 + x^M) + r$ for some q and q'. Let us suppose that f is a multiple of p. Since M is a multiple of $\mathrm{ord}(p)$, $1 + x^M$ is a multiple of p. Then r is also a multiple of p, and so f' is a multiple of p. □

Proposition 5.1.3. *Let $p = g_1 \cdots g_r$ with $\mathrm{GCD}(g_i, g_j) = 1$ for all $i \neq j$. We denote $N_i = ord(g_i)$ and $N = \mathrm{LCM}(N_1, \ldots, N_r)$. Given $w - 2$ distinct integers $0 < e_2 < \ldots < e_{w-1} < N$, if there exist r integers $e_w^{(1)}, \ldots, e_w^{(r)}$ with $0 < e_w^{(i)} < N_i$ such that for all i and j*

1. *$[0, e_2, \ldots, e_{w-1}, e_w^{(i)}]$ is a multiple of g_i*
2. *$e_w^{(i)} \equiv e_w^{(j)} \pmod{\mathrm{GCD}(N_i, N_j)}$*

then $[0, e_2, \ldots, e_{w-1}, e_w]$ *is a multiple of* p *of weight* w*, where*

$$e_w = \mathrm{CRT}(e_w^{(1)}, \ldots, e_w^{(r)}, N_1, \ldots, N_r).$$

Furthermore, all multiples of p *of weight* w *and degree at most* N *can be obtained in this way, that is, if* $[0, e_2, \ldots, e_{w-1}, e_w]$ *with* $0 < e_2 < \ldots < e_w < N$ *is multiple of* p *of weight* w*, then the integers* $e_w^{(i)} = e_w \bmod N_i$ *satisfy points 1 and 2 above.*

Proof. Let $j \in \{1, \ldots, r\}$ and let us prove that $[0, e_2, \ldots, e_{w-1}, e_w]$ is a multiple of g_j. By point 2 we can apply CRT and get an integer $e_w < N$ such that $e_w \equiv e_w^{(i)} \pmod{N_i}$ for all i. Then, by point 1, applying Proposition 5.1.2 to $f = [0, e_2, \ldots, e_{w-1}, e_w]$ and $f' = [0, e_2, \ldots, e_{w-1}, e_w^{(j)}]$, we get that f is a multiple of g_j.
Let $[0, e_2, \ldots, e_{w-1}, e_w]$ be a multiple of p of weight w of degree at most N, and for $i \in \{1, \ldots, r\}$ let $e_w^{(i)} = e_w \bmod N_i$. Then, for any i, $[0, e_2, \ldots, e_{w-1}, e_w]$ is a multiple of g_i, and so, by Proposition 5.1.2 the integers $e_w^{(i)}$'s satisfy point 1. On the other hand, thanks to CRT, since $e_w^{(i)} = e_w \bmod N_i$, they satisfy also point 2. □

Proposition 5.1.4 ([LN97, Lemma 2.12 (ii)])**.** *Let* p *be an irreducible polynomial and let* α *be a root of* p *in an extension field of* $\mathbb{F}_2$*. Then a polynomial* q *is a multiple of* p *if and only if* $q(\alpha) = 0$*.*

Remark 5.1.5. Let α be the root of a primitive polynomial p. Given $e_1, \ldots, e_{w-1}$, if we are able to compute

$$e_w = \mathrm{DLog}_\alpha(\alpha^{e_1} + \ldots + \alpha^{e_{w-1}}), \tag{5.2}$$

then, by Proposition 5.1.4, we know that $[e_1, \ldots, e_{w-1}, e_w]$ is a multiple of p. Viceversa, if $[e_1, \ldots, e_{w-1}, e_w]$ is a multiple of p, then (5.2) holds.

All the $(w-1)$-uples $0 \le e_1 < \ldots < e_{w-1} < N$ can be obtained by producing all the $(w-2)$-uples $0 \le e_1 < \ldots < e_{w-2} < N-1$ and appending an integer e_{w-1} with $e_{w-2} < e_{w-1} < N$. We want to show that in order to get a $(w-1)$-uple of exponents for which the expression (5.2) is computable, that is for which $\alpha^{e_1} + \ldots + \alpha^{e_{w-1}}$ is nonzero, we can take $0 \le e_1 < \ldots < e_{w-2} < N-1$ randomly and then choose e_{w-1} with $e_{w-2} < e_{w-1} < N$, when necessary avoiding a specific value depending on the values of $e_1, \ldots, e_{w-2}$.
Let us take $0 \le e_1 < \ldots < e_{w-2} < N$ randomly and let $\beta = \alpha^{e_1} + \ldots + \alpha^{e_{w-2}}$. If $\beta = 0$ then for any integer $0 \le e_{w-1} < N$ we have that $\alpha^{e_{w-1}}$ is never equal to β. In this case, for any integer e_{w-1} with $e_{w-2} < e_{w-1} < N$, the discrete logarithm (5.2) is computable and it is equal to e_{w-1}. Let us suppose that β is nonzero. Then there exist a unique integer $0 \le e < N$ such that $\alpha^e = \beta$. In this case for any integer $0 \le e_{w-1} < N$ with $e_{w-1} \neq e$ we have that $\alpha^{e_{w-1}} \neq \beta$. So, if $e \le e_{w-2}$ then the for any $e_{w-2} < e_{w-1} < N$ the logarithm (5.2) is computable. Otherwise it is computable for any $e_{w-2} < e_{w-1} < N$ with $e_{w-1} \neq e$.

We claim the following:

Proposition 5.1.6. *Let α be a root of a primitive polynomial p of order N. The set of $(w-2)$-uples of integers $e_2, \ldots, e_{w-1}$ such that $0 < e_2 < \ldots < e_{w-1} < N$ has size $\binom{N-1}{w-2}$. Its subset such that we can compute $\mathrm{DLog}_\alpha(1 + \alpha^{e_2} + \ldots + \alpha^{e_{w-1}})$ has size at least $\binom{N-2}{w-2}$.*

Proof. The number of $(w-2)$-uples of integers $e_2, \ldots, e_{w-1}$ such that $0 < e_2 < \ldots < e_{w-1} < N$, is equal to the number of $(w-2)$-combinations that can be drawn from a set of $N-1$ elements, which is $\binom{N-1}{w-2}$, as claimed.

Let A be the set of $(w-2)$-uples of integers $e_2, \ldots, e_{w-1}$ with $0 < e_2 < \ldots < e_{w-1} < N$ for which we can compute $\mathrm{DLog}_\alpha(1+\alpha^{e_2}+\ldots+\alpha^{e_{w-1}})$ and let us prove that $|A| \geq \binom{N-2}{w-2}$. We showed that to obtain an element of A, it is enough to produce a $(w-2)$-uple $[0, e_2, \ldots, e_{w-2}]$ with $0 < e_2 < \ldots < e_{w-2} < N-1$, and choose e_{w-1} with $e_{w-2} < e_{w-1} < N$ avoiding the value $e = \mathrm{DLog}_\alpha(\alpha^{e_1} + \ldots + \alpha^{e_{w-2}})$ if $e > e_{w-2}$. Since in this process the $(w-2)$-uples that we exclude are at most $\binom{N-2}{w-3}$, the size of the set A is at least

$$\binom{N-1}{w-2} - \binom{N-2}{w-3} = \binom{N-2}{w-2}$$

□

Note that for small w and large N, $\frac{\binom{N-2}{w-2}}{\binom{N-1}{w-2}} = \frac{(N-w+1)}{(N-1)} \simeq 1$. Then, for almost all $(e_2, \ldots, e_{w-1})$ we can compute $\mathrm{DLog}_\alpha(1+\alpha^{e_2}+\ldots+\alpha^{e_{w-1}})$. Using this remark, in our algorithm we produce such $(w-1)$-uple taking $0 \leq e_2 < \ldots < e_{w-1} < N$ randomly and then checking if the e_{w-1} is such that the logarithm (5.2) is computable. If not, we iterate the process.

Lemma 5.1.7. *Let p be an irreducible non-primitive polynomial of degree n and order N, and let $m = (2^n - 1)/N$. Then there exists a primitive element $\tilde{\alpha}$ in $\mathbb{F}_{2^n}$ such that $p(\tilde{\alpha}^m) = 0$.*

Proof. Let $\beta \in \mathbb{F}_{2^n}$ be a root of p and let α be a primitive element of $\mathbb{F}_{2^n}$. Then, there exists a unique integer $M < 2^n - 1$ such that $\beta = \alpha^M$. Since β has order N, then $\beta^N = (\alpha^M)^N = 1$. So, M is a multiple of m, because α has order $2^n - 1$ and $(2^n - 1) = mN$. Let $k = M/m$. Since α has order $2^n - 1$, then α^m has order N. So, both $\beta = \alpha^{km}$ and α^m have order N. This implies that $\mathrm{GCD}(k, N) = 1$. If the integer k is such that $\mathrm{GCD}(k, 2^n - 1) = 1$ then α^k is a primitive element. So, in this case, the statement is true with $\tilde{\alpha} = \alpha^k$, since $\beta = (\alpha^k)^m$.
Otherwise, $\mathrm{GCD}(k, 2^n - 1) > 1$. We claim that in this case there exists an integer λ with $0 < \lambda < m$ such that $\mathrm{GCD}(k + \lambda N, m) = 1$. Since $\mathrm{GCD}(k, N) = 1$, then $\alpha^{(k+\lambda N)}$ is a primitive element, but $\alpha^{(k+\lambda N)m} = \alpha^{km} = \beta$, so we can choose $\tilde{\alpha} = \alpha^{(k+\lambda N)}$.

We are now left with proving our claim on the existence of such an integer λ. Let λ be the greatest divisor of m such that $\mathrm{GCD}(\lambda, k) = 1$. By definition of λ, for any prime which divides m, either it divides λ or (exclusively) it divides k. Suppose, for the sake of contradiction, that there exists a prime p which divides both m and $k + \lambda N$. Since p divides m, then one of the following cases occurs:

$\boldsymbol{p \mid \lambda}$: Then $p \nmid k$, and so, $p \nmid k + \lambda N$, which contradicts the choice of p.

$\boldsymbol{p \mid k}$: Then $p \nmid \lambda$. Moreover, since $\mathrm{GCD}(k, N) = 1$, then $p \nmid N$. So $p \nmid k + \lambda N$, which is again a contradiction.

□

Proposition 5.1.8. *Let p be an irreducible non-primitive polynomial of degree n and order N, let $m = (2^n - 1)/N$ and let α be a primitive element in $\mathbb{F}_{2^n}$ such that α^m is a root of p (the existence of α is provided by Lemma 5.1.7). Then, f is a multiple of p with $f(0) \neq 0$ if and only if there exist integers $d_2, \ldots, d_{w-1}, d_w$ multiples of m such that $0 < d_2 < \cdots < d_{w-1} < 2^n - 1$, $d_w = \mathrm{DLog}_\alpha(1 + \alpha^{d_2} + \cdots + \alpha^{d_{w-1}})$ is computable and is a multiple of m, and $f = [0, e_2, \ldots, e_{w-1}, e_w]$ with $e_j = d_j/m$.*

Proof. Let $f = [0, e_2, \ldots, e_{w-1}, e_w]$ with $e_2, \ldots, e_w$ as in the statement and let us prove that $f(\alpha^m) = 0$. Since α^m is a root of p, by Proposition 5.1.4 this is enough to guarantee that f is a multiple of p. On the other hand, $f(\alpha^m) = 1 + ((\alpha^m)^{e_2} + \ldots + (\alpha^m)^{e_{w-1}} + (\alpha^m)^{e_w} = 1 + \alpha^{d_2} + \ldots \alpha^{d_{w-1}} + \alpha^{d_w} = 0$. The last equality follows from the hypothesis $d_w = \mathrm{DLog}_\alpha(1 + \alpha_2^d + \cdots + \alpha_{w-1}^d)$. Vice versa, let f of weight w. Suppose that $f = [0, e_2, \ldots, e_{w-1}, e_w]$ with $0 < e_2 < \cdots < e_{w-1} < N$. If f is a multiple of p, then, by Proposition 5.1.4, $f(\alpha^m) = 0$. From $f(\alpha^m) = 0$, we get that $e_j = d_j/m$ for all j and $d_w = \mathrm{DLog}_\alpha(1 + \alpha^{d_2} + \cdots + \alpha^{d_{w-1}})$. □

Remark 5.1.9. If $[e_1, \ldots, e_w]$ is multiple of p, then $[2^t e_1, \ldots, 2^t e_w]$ is multiple of p^b for all $b \leq 2^t$.

In Algorithm 3 we will use Remark 5.1.9 to find multiples of powers of irreducibles. Not all multiples can be found in this way unless the exponent for the repeated factor is a power of two. In this case a complete characterization of multiples f of power of irreducibles can be given using the polynomial $\mathrm{GCD}(f, Df)$. This is not convenient in our case, since $\mathrm{GCD}(f, Df)$ has generally a much higher weight than the weight of f.

5.2 The algorithm

In this section we propose an algorithm to solve Problem 1 making reference to its pseudocode, then we estimate its complexity and compare it to other approaches.

5.2.1 Description of Algorithm 3

We describe our proposed algorithm making reference to the pseudocode in Algorithm 3.

We take as input the factorization of the polynomial p:

$$p = p_1^{b_1} \cdot \ldots \cdot p_r^{b_r},$$

where $p_1, \ldots, p_r$ are irreducible polynomials and $b_1, \ldots, b_r$ positive integers. Computing the factorization is not computationally expensive, it can be efficiently computed using a probabilistic algorithm such as the Cantor-Zassenhaus algorithm (cfr [LN97]) and in some cases,

such as in correlation attacks to LFSR-based stream ciphers, the polynomial we are interested in is given already by its irreducible factors.

We start (line 2) by computing a root $\alpha_i^{m_i}$ for each irreducible polynomial p_i, expressed as a power of a primitive root α_i of $\mathbb{F}_{2^{\deg(p_i)}}$ (if the polynomial is primitive $m_i = 1$). Next (line 3) we compute the order of each root $\alpha_i^{m_i}$, and we set t as an integer (line 4) that will allow us to take into account powers of irreducible polynomials (in line 12 we will output the multiple of p as a multiple of $p_1 \cdot \ldots \cdot p_r$ elevated to 2^t).

Algorithm 3 Given the factorization of a polynomial p, w and D, the algorithm finds (if it exists) a multiple of p of weight w and degree at most D.

1: **function** $(p_1, \ldots, p_r, b_1, \ldots, b_r, w, D)$
▷ All lines with i repeat for $i = 1, \ldots, r$
2: $\alpha_i, m_i \leftarrow \mathrm{PrimitiveRoot}(p_i)$
▷ If p_i is primitive then $m_i = 1$
▷ If p_i is irreducible not primitive then $\alpha_i^{m_i}$ is a root of p_i
3: $N_i \leftarrow (2^{\deg(p_i)} - 1)/m_i$
▷ N_i is the order of $\alpha_i^{m_i}$
4: $t \leftarrow \mathrm{MinIntegerGreaterOrEqualThan}(\log_2 b_1, \ldots, \log_2 b_r)$
5: **repeat**
6: $e_2, \ldots, e_{w-1} \leftarrow \mathrm{RandomDistinctMultiplesLessThan}(D/2^t)$
▷ Random sampling of $(w-2)$ distinct integers $\leq D/2^t$
7: $e_{i,2}, \ldots, e_{i,k(i)} \leftarrow \mathrm{ReduceAndShorten}(e_2, \ldots, e_{w-1}, N_i)$
▷ Reduce modulo N_i and eliminate pairs of equal integers
▷ we might obtain a shorter sequence of integers ($k(i) \leq w-1$)
8: $e_{i,w} \leftarrow \mathrm{DLog}_{\alpha_i}(1 + \alpha_i^{m_i e_{i,2}} + \cdots + \alpha_i^{m_i e_{i,k(i)}})$
▷ If not possible, restart the cycle
▷ also, restart the cycle if $e_{i,w} \bmod m_i \neq 0$
9: $e_{i,w} \leftarrow \mathrm{Reduce}(e_{i,w}/m_i, N_i)$
10: $e_w \leftarrow \mathrm{CRT}(e_{1,w}, \ldots, e_{r,w}, N_1, \ldots, N_r)$
▷ If not possible, restart the cycle
11: **until** $2^t e_w \leq D$
12: **return** $[0, 2^t e_2, \ldots, 2^t e_w]$
13: **end function**

The main cycle samples $w-2$ distinct integers less than $D/2^t$ (line 6), then computes for each i their remainders modulo N_i and discards pairs of equal elements (line 7). The most demanding computation is done on line 8 where we compute the discrete logarithms $\mathrm{DLog}_{\alpha_i}(1+\alpha_i^{e_{i,2}} + \cdots + \alpha_i^{e_{i,k(i)}})$. If we are not able to perform this computation or one of the $e_{i,w}$'s is not a multiple of m_i, then we restart the cycle. Otherwise, we try to compute the exponent e_w through

the Chinese Remainder Theorem (line 10) given the exponents computed at previous step. If all N_i's are coprime we are sure to be able to perform this step, otherwise it might happen that a couple $(e_{i,w}, e_{j,w})$ is not congruent modulo $\mathrm{GCD}(N_i, N_j)$. In this case we restart the cycle. We remark that in the case of N_i not all coprimes it may happen that we are *never* able to compute this CRT. We will comment on this aspect in 5.2.1. The exit condition for the cycle (line 11) is verified if we have found a multiple of $p_1 \cdot \ldots \cdot p_r$ with degree at most $D/2^t$, from which we are easily able (line 12) to produce a multiple of p of degree at most D (which will have weight w by construction). The correctness of Algorithm 3 follows from the results cited in Section 5.1.

The case of N_is not all coprimes

Suppose p is the product of p_1, p_2 with $N_1 = \mathrm{ord}(p_1), N_2 = \mathrm{ord}(p_2)$ and that $\mathrm{GCD}(N_1, N_2)$ is not 1. In this case it may happen that we are never able to compute $\mathrm{CRT}(e_{1,w}, e_{2,w}, N_1, N_2)$, that is, all possible pairs $(e_{1,w}, e_{2,w})$ are not congruent modulo $\mathrm{GCD}(N_1, N_2)$. Using bounds on the minimal distance of cyclic codes coming from coding theory (see [PBH98, Vol 1, p.60 and following]), one can verify whether or not the CRT can be computed. If the minimal distance of the cyclic code generated by p is $\geq w_0$, then obviously the CRT cannot be computed for all $w < w_0$.

5.2.2 Complexity estimates

Let us denote with q the polynomial $p_1 \cdots p_r$. The complexity of Algorithm 3 is estimated in terms of the order N of the polynomial q and in terms of the target degree D under an appropriate Statistical Assumption (Subsection 5.2.2). Then (Subsection 5.2.2), we compare it to the complexity of birthday-based approaches expressing our estimates in terms of the complexity parameter n, the degree of q.

Complexity of Algorithm 3

We set n, the degree of polynomial q, to be our complexity parameter. Clearly, the main computational cost in the cycle is the computation of the discrete logarithms (line 8). We will restrict our complexity calculations to the cases where we can consider the computation of discrete logarithm as an $O(1)$ computation, by precomputing logarithm tables. This depends on the smoothness of the degrees δ_i of the irreducible factors of q, and we call a δ_i feasible if we can consider the computation of discrete logarithm in $\mathbb{F}_{2^{\delta_i}}$ as $O(1)$ and the precomputed tables do not require excessive storage in memory. Following [DLC07], we can consider all δ_i below 78 as feasible except for $\{37, 41, 49, 59, 61, 62, 65, 67, 69, 71, 74, 77\}$.

We are left to estimate the number of cycles which Algorithm 3 needs to go through before stopping. In order to do that, we make the following

Assumption 5.2.1. *All Discrete Logarithm computations* DLog_α *with random input produce an output which is uniformly random distributed over* $\{1, \ldots, M-1\}$*, where* M *is the order of*

α. *We assume also that for* $\alpha \neq \beta$ *the outputs of* DLog_α *and* $DLog_\beta$ *are independent random variables.*

Although the Statistical Assumption 1 is reasonable (and experimentally verifiable), there is an unfortunate case when there is a pair (i, j) where $\mathrm{GCD}(N_i, N_j) \neq 1$ and there exist no multiple of $p_i \cdot p_j$ of weight w (see discussion in Subsection 5.2.1).

Under this assumption, we can compute the $e_{i,w}$'s once every m_i cycles (i.e. always if $m_i = 1$). Furthermore, the $e_{i,w}$'s are uniformly distributed over $\{1, 2, \ldots, N_i - 1\}$ and so, by the Chinese Remainder Theorem, e_w can be computed with a probability P where

$$P = \frac{N}{\Pi_{i=1}^{r} N_i},$$

with $N := \mathrm{LCM}(\{N_i\}_{i=1,\ldots,r})$, and it will be uniformly distibuted over $\{1, 2, \ldots, N - 1\}$. Indeed, the Chinese Remainder Theorem guarantees a bijection between the set $\{(e_{1,w}, \ldots, e_{r,w}) \mid e_{i,w} \equiv e_{j,w} \mod \mathrm{GCD}(N_i, N_j), 1 \leq i < j \leq r\}$ and the integer set $\{1, \ldots, N - 1\}$ (cfr [Knu81, p.256]).

The condition that $2^t e_w \leq D$ is thus verified after a number of cycles estimated as

$$O\left(\frac{1}{P}\Pi_i m_i \cdot \frac{2^t N}{D}\right)$$

and the full complexity is estimated as

$$O\left(r\frac{1}{P}\Pi_i m_i \cdot \frac{2^t N}{D}\right). \tag{5.3}$$

We note that it is possible to consider P as constant with respect to n, and that $2^t N$ can be approximated by 2^n. In the next subsubsection we will express D as a function of n and compare estimate (5.3) with the complexity of birthday-based approaches.

Comparison with other approaches

Birthday-based approaches compute their complexities under a different statistical assumption:

Assumption 5.2.2. *The multiples of degree at most* D *of a polynomial of degree* n *has weight* w *with probability approximately* $\binom{D}{w-1}2^{-D}$.

Using this assumption, one can prove ([Gol96]) that the expected number of the polynomials multiples of weight w for large D is:

$$N_{n,w} \simeq \frac{\binom{D}{w-1}}{2^n} \simeq \frac{D^{w-1}}{(w-1)!2^n} \tag{5.4}$$

	Birthday			Generalized Birthday			Algorithm 3				
w	3	4	5	3	4	5	**3**	**4**	**4**	5	5
D	$2^{n/2}$	$2^{n/3}$	$2^{n/4}$	$2^{n/2}$	$2^{n/2}$	$2^{n/3}$	$2^{n/2}$	$2^{n/3}$	$2^{n/2}$	$2^{n/4}$	$2^{n/3}$
time	$2^{n/2}$	$2^{2n/3}$	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$	$2^{n/3}$	$2^{n/2}$	$2^{2n/3}$	$2^{n/2}$	$2^{3n/4}$	$2^{2n/3}$
memory	$2^{n/2}$	$2^{2n/3}$	$2^{n/4}$	$2^{n/2}$	$2^{n/2}$	$2^{n/3}$	1	1	1	1	1

Table 5.1: Comparing complexity of Algorithms. We compare only the part of complexity which is exponential in the degree of the polynomial of which we want to find the multiple. We set as D for comparison the D_0 and D_1 which are defined, respectively, by birthday and generalized birthday approaches. Algorithm 3 performs better for weigth 3 and for weight 4.

Using (5.4) one computes the (expected) critical degree where polynomials of degree w will start to appear as

$$D_0 \simeq (w-1)!^{\frac{1}{w-1}} \cdot 2^{\frac{n}{w-1}} \tag{5.5}$$

A method based on the birthday paradox ([MS89]) finds a multiple of weight w and minimal degree D_0 with time complexity

$$O(D_0^{\lceil \frac{w-1}{2} \rceil}), \tag{5.6}$$

and memory complexity

$$O(D_0^{\lfloor \frac{w-1}{2} \rfloor}). \tag{5.7}$$

Also, when it can be applied, Chose-Joux-Mitton algorithm ([CJM02]) allows to decrease the memory complexity to $O(D_0^{\lfloor \frac{w-1}{4} \rfloor})$.

Using generalized birthday problem ([Wag02]) one aims at finding a multiple of same weight but higher degree ($D_1 \simeq 2^{\frac{n}{1+\lfloor \log(w-1) \rfloor}}$) in less time. The time complexity for this approach is

$$O((w-1) \cdot D_1) \tag{5.8}$$

while memory complexity is $O(D_1)$.

In Table 5.1 we summarize the comparison of our discrete log approach with birthday based approaches. Birthday based approaches do not distinguish the polynomial p in terms of its factorization while Algorithm 3 does, so for our comparison we assumed to be in the situation of p a multiple of primitive polynomials whose degrees are coprime. To simplify we also kept out the constant terms and we compare only the part of complexity which depends strictly on n. We set as D for comparison the D_0 and D_1 which are defined — respectively — by birthday and generalized birthday approaches. We can see that Algorithm 3 performs better (same time complexity but much better memory complexity) than both birthday based approaches in the case of weights $w = 3, 4$ and it performs worse than both of them $w = 5$ (higher time complexity).

Previous discrete log based approaches were limited to p being a single primitive polynomial and cannot be extended straightforwardly to the case of general p.

In [KP94], Penzhorn and Kuhn described a method based on Zech's logarithm using a different statistical assumption on the output of Zech's Logarithms. They assumed that the difference of two Zech's Logarithms over $\{1, 2, \ldots, N-1\}$ has exponential distribution of parameter D/N when both inputs are randomly distributed over $\{1, 2, \ldots, D-1\}$. This is applied only for the case $w = 4$ and gives an algorithm which is able to compute a multiple of degree $2^{n/3}$ with time complexity $2^{n/3}$ outperforming all approaches mentioned above.

In [DLC07], Didier and Laigle-Chapuy used a discrete logarithm approach with time memory trade off to compute a polynomial multiple of degree D in time complexity $O(D^{\lceil \frac{w-2}{2} \rceil})$ and memory complexity $O(D^{\lfloor \frac{w-2}{2} \rfloor})$.

5.3 Significant Examples

In this section we consider two significant applications of Algorithm 3 to case of a correlation attack to E0 (Subsection 5.3) and to the case where the polynomial p can be decomposed in several irreducible factors (Subsection 5.3).

Example case: E0

In [LV04], Lu and Vaudenay describe a correlation attack to E0, the stream cipher of the Bluetooth protocol ([GPS04]). E0 is a nonlinear combination generator with memory and it is based on four LFSRs of degrees $25, 31, 33, 39$ with the following feedback polynomials:

$$p_1 = x^{25} + x^{20} + x^{12} + x^8 + 1 \tag{5.9}$$
$$p_2 = x^{31} + x^{24} + x^{16} + x^{12} + 1 \tag{5.10}$$
$$p_3 = x^{33} + x^{28} + x^{24} + x^4 + 1 \tag{5.11}$$
$$p_4 = x^{39} + x^{36} + x^{28} + x^4 + 1 \tag{5.12}$$

In the key-recovery attack of [LV04], two precomputation steps require to compute a weight-5 multiple of $p_2 \cdot p_3 \cdot p_4$ of degree at most $2^{34.3}$ using generalized birthday approach, and a weight-3 multiple of $p_3 \cdot p_4$ of degree at most 2^{36} using a standard birthday approach. To find the first multiple precomputation, Algorithm 3 would be inconvenient since it would require a higher time complexity. For the second multiple precomputation, Algorithm 3 could be useful since it requires approximately the same time complexity while having a much lower memory complexity. We remark that in [LV04] the memory complexity for the precomputation part is not explicitly stated.

Note also that all p_i's are primitive and that $\mathrm{GCD}(N_3, N_4) = 2^3 - 1 = 7 \neq 1$. We are in the case where some of the N_i's are not coprime and for this specific polynomials, it is experimentally verified that the CRT can be computed roughly once every seven times.

Experimental results: several irreducible factors

We apply Algorithm 3 to the case where the polynomial p can be decomposed in four irreducible factors and each factor has degree less than 20. As set of degrees we choose the set $S = \{11, 13, 15, 17, 19\}$.
In Table 5.2 we show some timing to find a polynomial multiple of weight 3 of the product of four random primitive polynomials of weight 5 and different degrees in the set S. The polynomials P_1, P_2, P_3 in the table denote the product of four random polynomials each of degree in S. In this case the algorithm is substantially better than the generalized birthday approach as

	w=3		
	P_1	P_2	P_3
n	60	60	64
expected D	2^{30}	2^{30}	2^{32}
experimental D	$2^{30.05}$	$2^{28.87}$	$2^{31.56}$
time	$3h9'$	$6h2'$	$4h42'$

Table 5.2: Experimental timing in the case of the product of four random primitive polynomials of weight 5 and degrees in $\{11, 13, 15, 17, 19\}$. The polynomials used in the table are $P_1 = (x^{11}+x^7+x^5+x^4+1)(x^{13}+x^{12}+x^9+x^3+1)(x^{17}+x^{15}+x^{14}+x^9+1)(x^{19}+x^{18}+x^{15}+x^{13}+1)$, $P_2 = (x^{13}+x^{12}+x^{10}+x^9+1)(x^{15}+x^{14}+x^{13}+x^{11}+1)(x^{17}+x^{16}+x^{15}+x^{14}+1)(x^{19}+x^{18}+x^{17}+x^{14}+1)$, and $P_3 = (x^{11}+x^{10}+x^9+x^7+1)(x^{13}+x^{12}+x^{10}+x^9+1)(x^{17}+x^{16}+x^{15}+x^{14}+1)(x^{19}+x^{18}+x^{17}+x^{14}+1)$.

its time complexity is approximately the same as the generalized birthday approach while its memory complexity is $O(1)$ instead of $2^{n/2}$. This is due to the fact that, once we have stored the table of discrete logarithms, the computation of discrete logarithms in $\mathbb{F}_{2^m}$ with $m < 20$ is very fast.

On the shape of the general error locator polynomial

This chapter focus primarily on some issues concerning the efficiency of bounded-distance decoding for cyclic codes. It is organized as follows. In Section 6.1 we present a new result on the structure of the general error locator polynomial for a classo of cyclic codes. In Section 6.2 some properties of this locator given in [OS07] are extended and it is shown how the previous result can be used to obtain a sparse representation of the general error locator polynomial for three of the five exceptional cases of Theorem 3.3.30. In the same section we also exhibit a general error locator polynomial for some infinite classes of binary cyclic codes with $t = 2$, including the other two remaining cases of Theorem 3.3.30. In Section 6.3 we provide a such locator for all binary cyclic codes with $t = 3$ and $n < 63$. A sparse representation is theoretically justified for all cases, except three. In the same section we show a general structure of this locator for some infinite classes of binary cyclic codes with $t = 3$, adding theoretical evidence to the sparsity of the locator for infinite classes of codes. Finally, in Section 6.4.2 we study the complexity of bounded-distance decoding of certain classes of cyclic codes. Some results of this section are conditioned to a conjecture and others hold unconditionally.

6.1 A new representation of the locator polynomial

In this section we give a new general result on the structure of the error locator polynomial for a class of cyclic codes over $\mathbb{F}_q$.

Sometimes we will deal with rational expressions of the kind $\frac{f}{g}$, with $f, g \in \mathbb{F}_q[x_1, \ldots, x_\ell]$ for some $\ell \geq 1$. When we evaluate this expression on any point $P \in \mathbb{F}_q^\ell$ it is possible that $g(P) = 0$. However, our rational expressions are evaluated only in points such that if $g(P) = 0$ then also $f(P) = 0$, and when this happens we always use the convention that $\frac{f(P)}{g(P)} = 0$.

Let $R_n = \{\alpha^i \mid i = 0, \ldots, n-1\}$. Let us denote with $T_{n,t}$ the following set

$$T_{n,t} = \{(\alpha^{l_1}, \ldots, \alpha^{l_\mu}, 0, \ldots, 0) \mid 0 \leq l_1 < \cdots < l_\mu < n,\, 0 \leq \mu \leq t\} \subset (R_n \cup \{0\})^t.$$

Let C be a cyclic code over $\mathbb{F}_q$, with length n and error correction capability t, defined by $S_C = \{i_1, \ldots, i_r\}$. where r is at least the number of primary syndromes and at most $n - k$. We will assume that $\mathcal{V} \subset \mathbb{F}_{q^m}^r$ and $X = (x_1, \ldots, x_r)$ where x_j, for $j \in \{1, \ldots r\}$, denotes (if not differently specified) the variable corresponding to the syndrome s_{i_j}, i.e. $x_j = z_1^{i_j} + \cdots + z_t^{i_j}$. The main result of this section is Theorem 6.1.3. It generalizes Theorem 3.3.35, which dealt with the case where the code could be defined by only one syndrome. In particular,

Theorem 6.1.3 provides a description of the shape of the coefficients of a general error locator polynomial for cyclic codes over $\mathbb{F}_q$. We recall that, by definition of general error locator polynomial, these coefficients are polynomials in the syndrome variables X, and that when they are evaluated at a correctable syndrome $\mathbf{s}$ corresponding to an error of weight $\mu \leq t$, they can be expressed as the elementary symmetric functions on the error locations $\bar{z}_1, \ldots, \bar{z}_\mu$ and zero (with multiplicity $t - \mu$), since the latters are the roots of the locator. So, by definition of elementary symmetric functions, they can then be expressed as elementary symmetric polynomials in μ variables on the $z_1, \ldots, z_\mu$. We will need the existence of a polynomial representation for arbitrary functions from $\mathbb{F}_q^n$ to $\mathbb{F}_q$. This is not unique and can be obtained in several ways. We recall that a standard way to get it, is given by Theorem 2.3.20. The next two lemmas clarify some links between syndromes and error locations which will be essential in our proof of Theorem 6.1.3.

Lemma 6.1.1. *Let $\sigma \in \mathbb{F}_q[y_1, \ldots, y_t]$ be a symmetric function. Then there exists $a \in \mathbb{F}_q[X]$ such that for $(\bar{x}_1, \ldots, \bar{x}_r) \in \mathcal{V}^\mu$*

$$a(\bar{x}_1, \ldots, \bar{x}_r) = \sigma(z_1, \ldots, z_\mu, 0 \ldots, 0)$$

with $z_1, \ldots z_\mu$ the error locations corresponding to $\bar{x}_1 \ldots, \bar{x}_r$.

Proof. We claim that the statement is obvious for elementary symmetric functions, as follows. Let $\sigma_1 \ldots \sigma_t$ be the elementary symmetric functions in $\mathbb{F}_q[y_1, \ldots, y_t]$. The existence of a general error locator polynomial for any cyclic code guarantees that, for any $1 \leq i \leq t$, for any σ_i there is $a_i \in \mathbb{F}_q[x_1, \ldots, x_r]$ such that for $(\bar{x}_1, \ldots, \bar{x}_r) \in \mathcal{V}^\mu$

$$a_i(\bar{x}_1, \ldots, \bar{x}_r) = \sigma_i(z_1, \ldots, z_\mu, 0 \ldots, 0)$$

with $z_1, \ldots z_\mu$ the error locations corresponding to $\bar{x}_1 \ldots, \bar{x}_r$.

For the more general case of any symmetric function $\sigma \in \mathbb{F}_q[y_1, \ldots, y_t]$, we need the fundamental theorem on symmetric functions, which shows the existence of a polynomial $H \in \mathbb{F}_q[y_1 \ldots, y_t]$ such that $\sigma(y_1, \ldots, y_t) = H(\sigma_1(y_1, \ldots, y_t), \ldots, \sigma_t(y_1 \ldots, y_t))$. We can define $a = H(a_1, \ldots, a_t) \in \mathbb{F}_q[X]$. So for $(\bar{x}_1, \ldots, \bar{x}_r) \in \mathcal{V}^\mu$ and the corresponding locations $z_1, \ldots z_\mu$, we have

$$\begin{aligned}\sigma(z_1, \ldots, z_\mu, 0, \ldots, 0) &= H(\sigma_1(z_1, \ldots, z_\mu, 0, \ldots, 0), \ldots, \sigma_t(z_1, \ldots, z_\mu, 0, \ldots, 0)) = \\ &= H(a_1(\bar{x}_1, \ldots, \bar{x}_r), \ldots, a_t(\bar{x}_1 \ldots, \bar{x}_r)) = a(\bar{x}_1, \ldots, \bar{x}_r)\,.\end{aligned}$$

□

Lemma 6.1.2. *Let $h \in \mathbb{F}_q[X]$ with $\deg_{x_i} h < q$ for all $i = 1, \ldots, r$, and $h(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_r) = 0$ for all $(\bar{x}_1, \ldots, \bar{x}_r) \in \mathbb{F}_q^r$ with $\bar{x}_1 \neq 0$. Let $l \in \mathbb{F}_q[X]$ and $g(x_2 \ldots, x_r) \in \mathbb{F}_q[x_2, \ldots, x_r]$ such that $h(X) = x_1 l(x_1, x_2, \ldots, x_r) + g(x_2, \ldots, x_r)$. Then*

$$h(X) = g(x_2, \ldots, x_r) \quad \textit{or} \quad h(X) = (1 - x_1^{q-1}) \cdot g(x_2, \ldots, x_r).$$

Proof. Clearly, for any h the two polynomials l and g are uniquely determined.
If $h(X) \in \mathbb{F}_q[x_2 \ldots, x_r]$, trivially, we have that $h(X) = g(x_2, \ldots, x_r)$. So we can suppose that $h(X) \notin \mathbb{F}_q[x_2 \ldots, x_r]$, and let us define the polynomial $\bar{h}(X) = (1 - x_1^{q-1}) \cdot g(x_2, \ldots, x_r)$. Note that $\deg_{x_i} \bar{h} < q$ for all $i = 1, \ldots, r$. We claim that $h = \bar{h}$. Since the degree w.r.t. each variable x_i of both the polynomials h and $\bar{h}$ is less than q, to prove our claim we suffice to show that $h(\hat{X}) = \bar{h}(\hat{X})$ for all $\hat{X} \in \mathbb{F}_q^r$. Let us distinguish the cases $\hat{x}_1 = 0$ and $\hat{x}_1 \neq 0$.
If $\hat{x}_1 = 0$ then $h(\hat{X}) = 0 \cdot l(0, \hat{x}_2, \ldots, \hat{x}_r) + g(\hat{x}_2, \ldots, \hat{x}_r) = g(\hat{x}_2, \ldots, \hat{x}_r)$ and $\bar{h}(\hat{X}) = (1 - 0) \cdot g(\hat{x}_2, \ldots, \hat{x}_r) = g(\hat{x}_2, \ldots, \hat{x}_r)$. So, in this case $h(\hat{X}) = \bar{h}(\hat{X})$.
Otherwise, let $\hat{x}_1 \neq 0$. By hypothesis, $h(\hat{X}) = 0$. On the other hand, $\bar{h}(\hat{X}) = (1 - 1) \cdot g(\hat{x}_2, \ldots, \hat{x}_r) = 0$. So, also in this case $h(\hat{X}) = \bar{h}(\hat{X})$. □

Theorem 6.1.3. *Let C be a cyclic code over $\mathbb{F}_q$, with length n and correction capability t, defined by $S_C = \{i_1, \ldots, i_r\}$. Let $\sigma \in \mathbb{F}_q[y_1, \ldots, y_t]$ be a symmetric homogeneous function of total degree δ, with δ a multiple of i_1, and let λ be a divisor of n. Then there exist polynomials $a \in \mathbb{F}_q[X]$, $g \in \mathbb{F}_q[x_2, \ldots, x_r]$, some non-negative integers $\delta_2, \ldots, \delta_r$ and some univariate polynomials $F_{h_2,\ldots,h_r} \in \mathbb{F}_q[y]$ such that for any $0 \leq \mu \leq t$, for any $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_r) \in \mathcal{V}^\mu$ and the corresponding error locations $z_1, \ldots, z_\mu$, we have*

$$a(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_r) = \sigma(z_1, \ldots, z_\mu, 0, \ldots, 0) \tag{6.1}$$

and

$$a(X) = x_1^{\delta/i_1} \sum_{h_r=0}^{\delta_r} \left(\left(\frac{x_r}{x_1^{i_r}} \right)^{h_r} \cdots \sum_{h_2=0}^{\delta_2} \left(\left(\frac{x_2}{x_1^{i_2}} \right)^{h_2} F_{h_2,\ldots,h_r}(x_1^\lambda) \right) \cdots \right) + (1 - x_1^{q^m-1}) \cdot g(x_2, \ldots, x_r)\,, \tag{6.2}$$

where $\delta_i \leq q^m - 1$, $\deg F_{h_2,\ldots,h_r} \leq (q^m - 1)/\lambda$.

Proof. We observe that (6.1) is immediate by Lemma 6.1.1. To prove (6.2) we first show the case $\bar{x}_1 \neq 0$ and then the general case.

Case $\bar{x}_1 \neq 0$
Let us consider the following map $A : \{X \in \mathcal{V} \mid x_1 \neq 0\} \to \mathbb{F}_{q^m}$ defined by

$$A(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_r) = \frac{\sigma(z_1, \ldots, z_\mu, 0, \ldots, 0)}{\bar{x}_1^{\delta/i_1}}, \tag{6.3}$$

where $(z_1, \ldots, z_\mu, 0, \ldots, 0)$ is the element of $T_{n,t}$ associated to the syndrome vector $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_r)$. We claim that A depends only on $(\bar{x}_1^\lambda, \bar{x}_2/\bar{x}_1^{i_2}, \ldots, \bar{x}_r/\bar{x}_1^{i_r})$. If our claim is true, then we have that

$$A(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_r) = f(\bar{x}_1^\lambda, \bar{x}_2/\bar{x}_1^{i_2}, \ldots, \bar{x}_r/\bar{x}_1^{i_r}) \tag{6.4}$$

for a function $f : \mathbb{F}_{q^m}^r \to \mathbb{F}_{q^m}$, and so, by Lemma 2.3.20, we can view f as a polynomial in $\mathbb{F}_{q^m}[x_1, \ldots, x_r]$. Since $\mathcal{V} \subset \mathbb{F}_{q^m}^r$, we can also view A as a (non-unique) polynomial $A(X) \in$

$\mathbb{F}_{q^m}[X]$. On the other hand, (6.3) and Lemma 6.1.1 show that $A(X)x_1^{\delta/i_1} \in \mathbb{F}_{q^m}[X]$ equals a polynomial $a \in \mathbb{F}_q[X]$ and so also $A(X)$ can be chosen in $\mathbb{F}_q[X]$. Therefore, by (6.4) also f can be chosen in $\mathbb{F}_q[X]$.
Let $\delta_2 = \deg_{x_2}(f), \ldots, \delta_r = \deg_{x_r}(f)$. Then, by collecting the powers of x_r in f, we will have $f = \sum_{h=0}^{\delta_r} x_r^h f_h$, for some f_h's, which are polynomials in $\mathbb{F}_q[x_1, \ldots, x_{r-1}]$. We observe that for any $2 \leq i \leq r-1$ we have that, for any $0 \leq h \leq \delta_r$, $\deg_{x_i}(f_h) \leq \deg_{x_i}(f) = \delta_i$ and there is at least one h such that $\deg_{x_i}(f_{\mathsf{h}}) = \delta_i$. We can repeat this argument on all f_h's by collecting powers of x_{r-1} and iterate on the other X variables, x_1 excluded, until we obviously obtain the following formal description

$$f(x_1, x_2, \ldots, x_r) \quad = \quad \sum_{h_r=0}^{\delta_r} x_r^{h_r} \sum_{h_{r-1}=0}^{\delta_{r-1}} x_{r-1}^{h_{r-1}} \cdots \sum_{h_2=0}^{\delta_2} x_2^{h_2} F_{h_2,\ldots,h_r}(x_1)\,, \tag{6.5}$$

where any $F_{h_2,\ldots,h_r}$ is a univariate polynomial in $\mathbb{F}_q[x_1]$.
From (6.3), (6.4) and (6.5) we directly obtain the restriction of (6.2) to the case $x_1 \neq 0$, considering that $x_j^{q^m} = x_j$ implies $(1 - x_j^{q^m-1}) = 0$, $\delta_i \leq q^m - 1$ and $\deg F_{h_1,\ldots h_r} \leq (q^m-1)/\lambda$.

We now prove our claim that gives (6.4). Let us take $(\tilde{x}_1, \ldots, \tilde{x}_r)$ and $(\bar{x}_1, \ldots, \bar{x}_r)$ such that $\tilde{x}_k^\lambda = \bar{x}_k^\lambda$, and $\tilde{x}_j/\tilde{x}_1^{i_j} = \bar{x}_j/\bar{x}_1^{i_j}$, for $j = 2, \ldots, r$.
The first relation implies

$$\tilde{x}_1 = \beta \bar{x}_1, \tag{6.6}$$

for some β such that $\beta^\lambda = 1$.
Substituting $\tilde{x}_1$ for $\beta\bar{x}_1$ in the second relation for $j = 2, \ldots, r$, we obtain

$$\frac{\tilde{x}_j}{\tilde{x}_1^{i_j}} = \frac{\bar{x}_j}{\bar{x}_1^{i_j}} \implies \frac{\tilde{x}_j}{(\beta\bar{x}_1)^{i_j}} = \frac{\bar{x}_j}{\bar{x}_1^{i_j}} \implies \tilde{x}_j = \beta^{i_j}\bar{x}_j. \tag{6.7}$$

Suppose that $(\tilde{x}_1, \ldots, \tilde{x}_r) \in \mathcal{V}^\mu$ and $(\bar{x}_1, \ldots, \bar{x}_r) \in \mathcal{V}^{\mu'}$, with $\mu, \mu' \leq t$. From (6.6), we get $\tilde{y}_1\tilde{z}_1^{i_1} + \cdots + \tilde{y}_\mu\tilde{z}_\mu^{i_1} = \beta(\bar{y}_1\bar{z}_1^{i_1} + \cdots + \bar{y}_{\mu'}\bar{z}_{\mu'}^{i_1}) = \beta\bar{y}_1\bar{z}_1^{i_1} + \cdots + \beta\bar{y}_{\mu'}\bar{z}_{\mu'}^{i_1}$, where $\bar{z}_i$'s and $\bar{y}_i$'s are the locations and the error values, respectively, associated to $(\bar{x}_1, \ldots, \bar{x}_r)$; and similarly for $\tilde{z}_i$'s and $\tilde{y}_i$'s. Also, from (6.7) we get $\tilde{y}_1\tilde{z}_1^{i_j} + \cdots + \tilde{y}_\mu\tilde{z}_\mu^{i_j} = \beta^{i_j}(\bar{y}_1\bar{z}_1^{i_j} + \cdots + \bar{y}_{\mu'}\bar{z}_{\mu'}^{i_j})$, for $j = 2, \ldots, r$.
Let us now take $\hat{y}_j = \bar{y}_j$ and $\hat{z}_j = \beta\bar{z}_j$, for $j = 1, \ldots, \mu'$. Since the $\hat{z}_j$ are distinct valid error locations (i.e. $\hat{z}_j^n = 1$, for $j = 1, \ldots, \mu'$) we have that their syndromes are

$$\begin{aligned} \hat{x}_j \quad &= \quad \hat{y}_1\hat{z}_1^{i_j} + \cdots + \hat{y}_{\mu'}\hat{z}_{\mu'}^{i_j} = \beta^{i_j}\bar{y}_1\bar{z}_1^{i_j} + \cdots + \beta^{i_j}\bar{y}_{\mu'}\bar{z}_{\mu'}^{i_j} = \\ &= \quad \beta^{i_j}(\bar{y}_1\bar{z}_1^{i_j} + \cdots + \bar{y}_{\mu'}\bar{z}_{\mu'}^{i_j}) = \tilde{y}_1\tilde{z}_1^{i_j} + \cdots + \tilde{y}_\mu\tilde{z}_\mu^{i_j} = \tilde{x}_j, \\ \text{for} \quad & \quad j = 1, \ldots, r\,. \end{aligned}$$

Hence $(\hat{x}_1, \ldots, \hat{x}_r) = (\tilde{x}_1, \ldots, \tilde{x}_r)$, which implies that their corresponding locations and values must be the same and unique, because $\mu, \mu' \leq t$. Therefore $\mu = \mu'$, $\{\tilde{z}_1, \ldots, \tilde{z}_\mu\} =$

$\{\beta\bar{z}_1, \ldots, \beta\bar{z}_\mu\}$, and $\{\tilde{y}_1, \ldots, \tilde{y}_\mu\} = \{\bar{y}_1, \ldots, \bar{y}_\mu\}$, from which, using the fact that σ is a symmetric homogeneous function of degree δ, we have

$$A(\tilde{x}_1, \ldots, \tilde{x}_r) = \frac{\sigma(\tilde{z}_1, \ldots, \tilde{z}_\mu, 0, \ldots, 0)}{(\tilde{y}_1\tilde{z}_1^{i_1} + \cdots + \tilde{y}_\mu\tilde{z}_\mu^{i_1})^{\delta/i_1}} = \frac{\sigma(\beta\bar{z}_1, \ldots, \beta\bar{z}_\mu, 0, \ldots, 0)}{(\bar{y}_1(\beta\bar{z}_1)^{i_1} + \cdots + \bar{y}_\mu(\beta\bar{z}_\mu)^{i_1})^{\delta/i_1}} =$$
$$\frac{\beta^\delta\sigma(\bar{z}_1, \ldots, \bar{z}_\mu, 0, \ldots, 0)}{\beta^\delta(\bar{y}_1\bar{z}_1^{i_1} + \cdots + \bar{y}_\mu\bar{z}_\mu^{i_1})^{\delta/i_1}} = \frac{\sigma(\bar{z}_1, \ldots, \bar{z}_\mu, 0, \ldots, 0)}{(\bar{y}_1\bar{z}_1^{i_1} + \cdots + \bar{y}_\mu\bar{z}_\mu^{i_1})^{\delta/i_1}} = A(\bar{x}_1, \ldots, \bar{x}_r)\,.$$

General case

Let us consider the map A and the polynomial a introduced in the case $\bar{x}_1 \neq 0$. Let us extend A to all points in $\mathbb{F}_{q^m}^r$ defining $A(\tilde{X}) = \tilde{x}_1^{\delta/i_1}a(\tilde{x}_1, \ldots, \tilde{x}_r)$ when $\tilde{X} \in \mathbb{F}_{q^m} \setminus \mathcal{V}$ with $\tilde{x}_1 \neq 0$, and $A(\tilde{X})$ any element in $\mathbb{F}_{q^m}$ when $\tilde{X} = (0, \tilde{x_2}, \ldots, \tilde{x_r}) \in \mathbb{F}_{q^m}$. Since A is a map from $\mathbb{F}_{q^m}^r$ to $\mathbb{F}_{q^m}$, it is a polynomial function and the associated polynomial $A(X)$ with $deg_{x_i}(A(x)) < q$, for all $i = 1, \ldots, r$, is unique. Now, let us consider the polynomial $h(X) = a(X) - x_1^{\delta/i_1}A(X)$. Thanks to what we proved in the case $x_1 \neq 0$, we have that $h(X)$ satisfies the hypothesis of Lemma 6.1.2. Since $h(X) \notin \mathbb{F}_{q^m}[x_2, \ldots, x_r]$, by Lemma 6.1.2, we have that $h(X) = (1 - x_1^{q^m}) \cdot g(x_2, \ldots, x_r)$ for some $g(x_2, \ldots, x_r) \in \mathbb{F}_{q^m}$. So $a(X) = x_1^{\delta/i_1}A(X) + (1 - x_1^{q^m}) \cdot g(x_2, \ldots, x_r)$. □

Corollary 6.1.4. *Let C be cyclic code over $\mathbb{F}_q$ as in Theorem 6.1.3. Then the coefficients of the general error locator polynomial can be written in the form given by the previous theorem.*

Corollary 6.1.5. *Let C be a code with $t = 2$ defined by $S_C = \{i_1, \ldots, i_r\}$, with $i_1 = 1$, and let $\mathcal{L} = z^2 + x_1z + b$ be a general error locator polynomial for C. If C is a primitive code, i.e. $n = q^m - 1$, then $b = x_1^2A$ with $A \in \mathbb{F}_q[x_2/x_1^{i_2}, \ldots, x_r/x_1^{i_r}]$.*

Proof. Since $t = 2$, x_1 is zero if and only if there are no errors. Then, applying the previous theorem to C, we get that $b = x_1^2A$ with $A \in \mathbb{F}_q[x_1^n, x_2/x_1^{i_2}, \ldots, x_r/x_1^{i_r}]$. On the other hand, since C is primitive, x_1^n is zero when x_1 is zero, and it is 1 when x_1 is not zero. So for $\mu \in \{1, 2\}$, $x_1^n = 1$ and $b = x_1^2\bar{A}$ with $\bar{A} = A|_{x_1^n=1}$. We claim that $b^* = x_1^2\bar{A}$ is a valid location product also for the case $\mu = 0$, which follows from the fact that $\mu = 0$ if and only if $x_1 = 0$. □

Previous corollary basically shows that in the case $t = 2$ the term of the form $(1-x_1^{q^m-1})g$ does not appear in the expression of the locator coefficients.

6.2 Sparse locators for some classes of codes with $t = 2$

In this section we provide explicit sparse representations for infinite classes of codes with $t = 2$, including all the five exceptional cases of Theorem 3.3.30. For some of these we also show the connection with Theorem 6.1.3. Let C be a cyclic code with length n and error

correction capability $t = 2$, and let S_C be a defining set of C. In Chapter 3 we have defined a bordering polynomial as a polynomial $\mathsf{h} \in \mathbb{F}_2[X]$ such that $\mathsf{h}(\mathcal{V}^1) = \{0\}$ and $\mathsf{h}(\mathcal{V}^2) = \{1\}$. If $0 \in \tilde{S}_C$, then, by Remark 3.3.28, it is trivial to exhibit a bordering polynomial. If $0 \notin S_C$, we claim that in order to find such an h it is sufficient to find a polynomial $\mathsf{wh} \in \mathbb{F}_2[X]$, expressed in terms of primary syndromes only, such that

$$\mathsf{wh}(\mathcal{V}^1) = \{0\} \quad \text{and} \quad 0 \notin \mathsf{wh}(\mathcal{V}^2),$$

and we will call it a **weakly bordering polynomial**.

Lemma 6.2.1. *Let* wh *be a weakly bordering polynomial for* C*. Then* $\mathsf{h} = \mathsf{wh}/\overline{\mathsf{wh}}$ *is a bordering polynomial for* C *where* $\overline{\mathsf{wh}}$ *is a rewriting of* wh *using non-primary syndromes.*

Proof. Let wh be a weakly bordering polynomial for C. Given a primary syndrome x_j occurring in wh, we can consider the syndrome x_i such that $x_j = x_i^2$, $2i \equiv j \pmod{n}$, and take $\overline{\mathsf{wh}}$ as the rewriting of wh obtained with the substitution $x_j \to x_i^2$. This allows us to consider $\mathsf{h} = \mathsf{wh}/\overline{\mathsf{wh}}$, with the usual convention $0/0 = 0$. It is obvious that such an h is a bordering polynomial for C. □

From now on, in order to write a general error locator polynomial for a binary cyclic code C with $t = 2$, we will be satisfied with exhibiting b^* (as in Proposition 3.3.29) and wh as above such that either $b^*(\mathcal{V}^0) = 0$ or $\mathsf{wh}(\mathcal{V}^0) = 0$.

The following obvious lemma characterizes s2ec codes with $1 \in S_C$ and it is a direct generalization of Lemma 3.3.33.

Lemma 6.2.2. *Let* C *be a code with length* n *and* $S_C = \{i_1, i_2, \ldots, i_r\}$*, with* $i_1 = 1$*. The following statements are equivalent:*

a) for any $\{\tilde{z}_1, \tilde{z}_2\}$ *and* $\{\bar{z}_1, \bar{z}_2\}$ *subsets of* R_n *such that*

$$\tilde{z}_1^{i_j} + \tilde{z}_2^{i_j} = \bar{z}_1^{i_j} + \bar{z}_2^{i_j} \quad j = 1, \ldots, r,$$

we have $\{\tilde{z}_1, \tilde{z}_2\} = \{\bar{z}_1, \bar{z}_2\}$

b) C *is a* s2ec code.

The following lemma is a generalization of Lemma 3.3.32.

Lemma 6.2.3. *Let* C *be a code with* $S_C = \{0, 1, l\}$ *with error correction capability* t*. Let* C' *be a code with the same length of* C *and* $S'_C = \{1, l\}$*. Then the error correction capability of* C *is* $t = 2$ *if and only if* C' *is* s2ec.

Proof. Suppose that $t = 2$. To prove that C' is *s2ec* we will use the previous lemma. So, let $\{\tilde{z}_1, \tilde{z}_2\}$ and $\{\bar{z}_1, \bar{z}_2\}$ be subsets of R_n such that $\tilde{z}_1 + \tilde{z}_2 = \bar{z}_1 + \bar{z}_2$ and $\tilde{z}_1^l + \tilde{z}_2^l = \bar{z}_1^l + \bar{z}_2^l$, and we shall prove that $\{\tilde{z}_1, \tilde{z}_2\} = \{\bar{z}_1, \bar{z}_2\}$. Since we are assuming that $t = 2$, C is obviously *s2ec*. Note also that $\tilde{z}_1^n + \tilde{z}_2^n = \bar{z}_1^n + \bar{z}_2^n = 0$, since $\tilde{z}_1$, $\tilde{z}_2$, $\bar{z}_1$, $\bar{z}_2$ are elements of R_n. So, applying the previous lemma to C, we get that $\{\tilde{z}_1, \tilde{z}_2\} = \{\bar{z}_1, \bar{z}_2\}$.
Suppose, vice-versa, that C' is *s2ec*. C is a subcode of C', then also C is *s2ec*. In addition, from the BCH bound we know that $t \geq 1$. Then we are reduced to proving that we are able to distinguish the case with one error to the case with two errors. Let x_0 be the syndrome corresponding to 0, that is, $x_0 = \bar{z}_1^n + \bar{z}_2^n$. When $\mu = 1$ then $x_0 = 1 + 0 = 1 \neq 0$, while when $\mu = 2$ then $x_0 = 1 + 1 = 0$. This proves that $t = 2$. □

Note that in [OS07] the authors proved a special case of Theorem 6.1.3: let C be a binary cyclic code with length n and defining set $S_C = \{1\}$, and $\mathcal{L}^*$ the error locator polynomial that is able to correct two errors, such that $\mathcal{L}^* = z^2 + x_1 z + b^*(x_1)$. Then there exists a polynomial $A \in \mathbb{F}_2[y]$, such that

$$b^*(x_1) = A(x_1^n)x_1^2 .$$

The first four exceptional cases have $t = 2$ and $S_C = \{1, l\}$, so we can apply Theorem 6.1.3 and get b^*. As regards the fifth case, we consider the code C', defined by $S_{C'} = \{1, 5\}$, instead of C defined by $\{0, 1, 5\}$, and note that C' is a s2ec (by Lemma 6.2.3). In this way we can get b^* for C'. Notice that b^* for C is the same as that for C', since the additional syndrome (corresponding to 0) plays merely the role of determining the error weight. In conclusion, the polynomial b^* can be determinated for three exceptional cases by Theorem 6.1.3. At this point we are still left with the problem of determining b, which reduces to finding h (or wh). Note that $b^*(\mathcal{V}^0) = 0$. In what follows we show how we can do this for two out of three exceptional cases of Theorem 3.3.30.

Theorem 6.2.4. *Let C be a binary code with $\{1, l\} \subset S_C$, such that $l = 2^v + 1$, $v \geq 1$, $t = 2$ and $\gcd(l-2, n) = 1$. Then we can take $\mathsf{wh} = x_1^l + x_2$, where x_1 is the syndrome corresponding to 1 and x_2 is the syndrome corresponding to l.*

Proof. We see that

$$\begin{aligned} x_1^l + x_2 \; &= (z_1 + z_2)^l + z_1^l + z_2^l = (z_1^{2^v} + z_2^{2^v})(z_1 + z_2) + z_1^l + z_2^l = \\ &= z_1^{2^v} z_2 + z_1 z_2^{2^v} = z_1 z_2 (z_1^{2^v - 1} + z_2^{2^v - 1}). \end{aligned}$$

We know that $z_1 z_2 \neq 0$, so we are left to prove $(z_1^{2^v-1} + z_2^{2^v-1}) \neq 0$. This holds because if $z_1^{2^v-1} = z_2^{2^v-1}$, then $\gcd(l-2, n) = 1$ implies $z_1 = z_2$, which is impossible. □

We can apply Theorem 6.2.4 to describe the cases $n = 31$, with $S_C = \{1, 5\}$, and $n = 51$, with $S_C = \{1, 9\}$, of Theorem 3.3.30, taking $l = 5$ and $v = 2$ and $l = 9, v = 3$, respectively. In order to get h for the last case treated by Theorem 6.1.3, i.e. $n = 51$, $S_C = \{0, 1, 5\}$, we use the syndrome x_0 corresponding to 0 and take $\mathsf{h} = x_0 + 1$ (as in Section VI of [OS07]).

There are still two cases left. These require a different approach. However, it turns out that this new approach can actually solve also the previous three cases in an even more efficient way.

Theorem 6.2.5. *Let C be a binary cyclic code with $\{1, l, s-2l, s-l, s\} \subset S_C$, $t = 2$, $\gcd(s-2l, n) = \gcd(l, n) = 1$. Let x_1,x_2,x_3,x_4,x_5 be the syndromes corresponding respectively to 1,l,$s-2l$, $s-l$, s. Then*

$$b = b^* = \left(\frac{x_2x_4 + x_5}{x_3}\right)^{l^+}, \quad \mathcal{L} = z^2 + x_1 z + b, \tag{6.8}$$

where l^+ is the inverse of l modulo n.

Proof. Let us consider x_4x_2:

$$\begin{aligned} x_4x_2 &= (z_1^{s-l} + z_2^{s-l})(z_1^l + z_2^l) = z_1^s + z_2^s + z_1^{s-l}z_2^l + z_1^l z_2^{s-l} = \\ &= z_1^s + z_2^s + (z_1z_2)^l(z_1^{s-2l} + z_2^{s-2l}) = x_5 + (b^*)^l x_3 \end{aligned}$$

Therefore we have $(b^*)^l = (x_2x_4 + x_5)/x_3$ and by the fact that $\gcd(l, n) = 1$ we have $b^* = ((x_2x_4 + x_5)/x_3)^{l^+}$.

For $\mu = 2$ we have $x_3 = z_1^{s-2l} + z_2^{s-2l}$ which cannot be zero because $\gcd(s-2l, n) = 1$.

We also have $b^* = b$, because $b^*(\mathcal{V}^0) = 0$ and for $\mu = 1$, $x_2x_4 + x_5 = z^l z^{s-l} + z^s = 0$.

□

Theorem 6.2.6. *Let C be a code with $\{1, (n-1)/l\} \subset S_C$, l a power of 2, $3 \nmid n$, $l \neq (n-1)$, and $t = 2$. Let x_1,x_2,x_3,x_4 be the syndromes corresponding to 1,2, $(n-1)/l$, $n-2$. Then*

$$b^* = \frac{x_1}{x_3^l}, \quad \mathsf{wh} = x_4x_1^2 + 1, \quad \mathcal{L} = z^2 + x_1 z + b^* \mathsf{wh}/\overline{\mathsf{wh}},$$

where $\overline{\mathsf{wh}} = x_4x_2 + 1$.

Proof. For $\mu = 2$ and by the fact that l is a power of 2 we have

$$\begin{aligned} x_1x_3^l &= (z_1 + z_2)(z_1^{(n-1)/l} + z_2^{(n-1)/l})^l = \\ &= (z_1 + z_2)(z_1^{n-1} + z_2^{n-1}) = z_1^n + z_1z_2^{-1} + z_2z_1^{-1} + z_2^n = \\ &= \frac{z_1}{z_2} + \frac{z_2}{z_1} = \frac{z_1^2 + z_2^2}{z_1z_2} = \frac{(z_1 + z_2)^2}{b^*} = x_1^2/b^* \end{aligned}$$

from which we get $b^* = x_1x_3^{-l}$.

Let us now consider $\mathsf{wh}(x_1, x_2, x_3, x_4) = x_4x_1^2 + 1$. Since $b^*(\mathcal{V}^0) = 0$, we only need to prove that $\mathsf{wh}(\mathcal{V}_1) = 0$ and $0 \notin \mathsf{wh}(\mathcal{V}_2)$. For $\mu = 1$ we have $\mathsf{wh} = z^{n-2}z^2 + 1 = z^n + 1 = 0$.

For $\mu = 2$ we have

$$\begin{gathered} (z_1^{n-2} + z_2^{n-2})(z_1^2 + z_2^2) + 1 = z_1^{n-2}z_2^2 + z_2^{n-2}z_1^2 + 1 = \\ \frac{z_2^2}{z_1^2} + \frac{z_1^2}{z_2^2} + 1 = \left(\frac{z_2}{z_1} + \frac{z_1}{z_2}\right)^2 + 1. \end{gathered}$$

We are then left with the problem of proving

$$\frac{z_2}{z_1} + \frac{z_1}{z_2} \neq 1.$$

This is equivalent to $\alpha + \alpha^{-1} \neq 1$ for $\alpha = \frac{z_2}{z_1}$. By contradiction, if we multiply by α we get $\alpha^2 + \alpha + 1 = 0$, which implies $\alpha \in \mathbb{F}_4 \setminus \mathbb{F}_2$, from which follows that the order of α must be 3. However, $\alpha^n = 1$ (since z_1 and z_2 are error locations) and this is impossible because we have assumed $3 \nmid n$. □

Theorem 6.2.7. *Let C be a code with $\{1, l = 2^j, r = 2^j - 2^i, s = 2^j + 2^{(i+1)}\} \subset S_C$ with $i \geq 0$ and $j \geq i + 2$, and let $t = 2$. Let x_1,x_2,x_3,x_4 be the syndromes corresponding respectively to 1, l, r, s. Then*

$$(x_2 + x_1^{l-r} x_3) b^{*l-r} = (x_1^s + x_4)$$

Proof. We claim that in $\mathbb{F}_2[z_1, z_2]$ for all $i, j \in \mathbb{N}_0$, $j \geq i + 2$:

$$(z_1 + z_2)^{2^j + 2^{(i+1)}} + z_1^{2^j + 2^{(i+1)}} + z_2^{2^j + 2^{(i+1)}} = (z_1 z_2)^{2^i} \left(z_1^{2^j} + z_2^{2^j} + (z_1 + z_2)^{2^i} (z_1^{2^j - 2^i} + z_2^{2^j - 2^i}) \right)$$

The statement is an immediate consequence of this claim.

We now prove our claim. The right-hand side of the equality we want to prove is equal to

$$(z_1 z_2)^{2^i} \left(z_1^{2^j} + z_2^{2^j} + (z_1^{2^i} + z_2^{2^i})(z_1^{2^j - 2^i} + z_2^{2^j - 2^i}) \right) =$$
$$(z_1 z_2)^{2^i} \left(z_1^{2^i} z_2^{2^j - 2^i} + z_1^{2^j - 2^i} z_2^{2^i} \right) = z_1^{2^{(i+1)}} z_2^{2^j} + z_1^{2^j} z_2^{2^{(i+1)}}$$

On the other hand, since $(z_1 + z_2)^{2^{(i+1)} + 2^j} = (z_1 + z_2)^{2^{(i+1)}} (z_1 + z_2)^{2^j} = (z_1^{2^{(i+1)}} + z_2^{2^{(i+1)}})(z_1^{2^j} + z_2^{2^j})$, also the left-hand side is equal to $z_1^{2^{(i+1)}} z_2^{2^j} + z_1^{2^j} z_2^{2^{(i+1)}}$. □

Corollary 6.2.8. *With the same hypothesis as in Theorem 6.2.7, and the assumptions that $i = 0$ and $\gcd(r - 1, n) = 1$ we have that*

$$b = b^* = \left(\frac{x_1^s + x_4}{x_2 + x_1 x_3} \right), \quad \mathcal{L} = z^2 + x_1 z + b,$$

Proof. By Theorem 6.2.7, we get that

$$(x_2 + x_1^{l-r} x_3) b^{*l-r} = (x_1^s + x_4).$$

Let us suppose that $i = 0$ and $\gcd(r - 1, n) = 1$. Then, the previous equality becomes

$$(x_2 + x_1 x_3) b^* = (x_1^s + x_4). \tag{6.9}$$

First we show that $\mu = 2$ implies $(x_2 + x_1 x_3) \neq 0$. We have that

$$x_2 + x_1 x_3 = (z_1^{2^j} + z_2^{2^j}) + (z_1 + z_2)(z_1^{2^j - 1} + z_2^{2^j - 1}) =$$
$$(z_1^{2^j} + z_2^{2^j}) + (z_1^{2^j} + z_2^{2^j} + z_1 z_2^{2^j - 1} + z_1^{2^j - 1} z_2) = z_1 z_2 (z_1^{2^j - 2} + z_2^{2^j - 2}).$$

Now, because $\mu = 2$ and $\gcd(r-1, n) = \gcd(2^j - 2, n) = 1$, $z_1 z_2 (z_1^{2^j-2} + z_2^{2^j-2}) \neq 0$. Then $(x_2 + x_1 x_3)$ is nonzero. So, we get $b^* = (x_1^s + x_4)/(x_2 + x_1 x_3)$.
Furthermore, since $b^*(\mathcal{V}^0) = 0$ and for $\mu = 1$, $x_1^s + x_4 = z^s + z^s = 0$, we also conclude that $b^* = b$. □

To obtain an efficient general error locator polynomial for the code with length $n = 51$ and defining set $S_C = \{0, 1, 5\}$ we will need the following result.

Lemma 6.2.9. *Let C be the code with length $n = 51$ defined by $S_C = \{0, 1, 5\}$, and let $\mu = 2$. Denoting by x_j the variable corresponding to the syndrome s_j, i.e. $x_j = z_1^j + z_2^j$, and by b the product $z_1 z_2$, we have that*

$$(x_1^7 + x_7) = 0 \quad \textit{if and only if} \quad x_1^{51} = 1.$$

Proof. Let us suppose that $(x_1^7 + x_7) = 0$. We have that

$$\begin{aligned} &x_1^7 + x_7 = (z_1 + z_2)^7 + z_1^7 + z_2^7 = z_1^6 z_2 + z_1^5 z_2^2 + z_1^4 z_2^3 + z_1^3 z_2^4 + z_1^2 z_2^5 + z_1 z_2^6 = \\ &(z_1 z_2)(z_1^5 + z_2^5 + z_1^4 z_2 + z_1 z_2^4 + z_1^2 z_2^3 + z_1^3 z_2^2) = (z_1 z_2)\left((z_1 + z_2)^5 + (z_1 z_2)^2 (z_1 + z_2)\right). \end{aligned} \tag{6.10}$$

Since $(x_1^7 + x_7) = 0$, by (6.10), we have that $(z_1 z_2)\left((z_1 + z_2)^5 + (z_1 z_2)^2 (z_1 + z_2)\right) = 0$. But $z_1 z_2 \neq 0$, because $\mu = 2$. Thus, $(z_1 + z_2)^5 = (z_1 z_2)^2 (z_1 + z_2)$. Then

$$\left((z_1 + z_2)^5\right)^{51} = \left((z_1 z_2)^2 (z_1 + z_2)\right)^{51}. \tag{6.11}$$

Since the splitting field of $x^{51} + 1$ over $\mathbb{F}_2$ is $\mathbb{F}_{256}$, then $x_1^{255} = 1$. But we have also that $z_1 z_2 \in \mathbb{F}_{256}$, so $(z_1 z_2)^{2 \times 51} = 1$. Then, by (6.11) we get that $(z_1 + z_2)^{51} = 1$.

Vice-versa, if $x_1^{51} = 1$ then $x_1^{64} = x_1^{13}$. But $x_1^{64} = z_1^{64} + z_2^{64} = z_1^{13} + z_2^{13} = x_{13}$, so

$$x_1^{13} = x_{13}. \tag{6.12}$$

By Newton's identities (see Theorem 2.3.27), we know that $x_1^5 = x_5 + b x_3$. Then,

$$\begin{aligned} &x_1^{13} = x_1^5 x_1^8 = (x_5 + b x_3) x_1^8 = x_5 x_1^8 + b x_3 x_1^8 = (z_1^5 + z_2^5)(z_1^8 + z_2^8) + b x_3 x_1^8 = \\ &z_1^{13} + z_2^{13} + z_1^5 z_2^8 + z_1^8 z_2^5 + b x_3 x_1^8 = x_{13} + b^5 x_3 + b x_3 x_1^8. \end{aligned} \tag{6.13}$$

By (6.13) and (6.12), we obtain that $b^5 x_3 + b x_1^8 x_3 = 0$. Thus, since $b \neq 0$, either $b = x_1^2$ or $x_3 = 0$.
If $b = x_1^2$, then, by (6.10), $x_1^7 + x_7 = x_1^2 (x_1^5 + x_1^4 x_1) = 0$.
If $x_3 = 0$ then $x_1^5 = x_5 + b x_3 = x_5$. Then

$$x_1^7 = x_1^5 x_1^2 = x_5 x_1^2 = (z_1^5 + z_2^5)^5 (z_1^2 + z_2^2) = x_7 + b x_3. \tag{6.14}$$

Since $x_3 = 0$, by (6.14), we obtain that $x_1^7 = x_7$. □

We now apply the previous results to the exceptional cases of Theorem 3.3.30. Summing up we get the following.

a) Case $\mathbf{n = 31, S_C = \{1, 15\}}$

This is a special case of those covered by Theorem 6.2.6 for $l = 1$, i.e. where x_1, x_2, x_3, x_4 correspond to $1, 2, 30, 29$. Hence we have $b^* = x_1/x_2$, $\mathsf{wh} = x_1^2 x_4 + 1$, and so the general error locator polynomial for the code is

$$z^2 + x_1 z + \frac{x_1}{x_3}\left(\frac{x_1^2 x_4 + 1}{x_2 x_4 + 1}\right), \tag{6.15}$$

b) Case $\mathbf{n = 31, S_C = \{1, 5\}}$

This case is a special case of Theorem 6.2.5 for $l = 1$ and $s = 10$. Therefore we have that the general error locator polynomial for C is

$$z^2 + x_1 z + \left(\frac{x_1 x_4 + x_5}{x_3}\right), \tag{6.16}$$

where x_1, x_3, x_4, x_5 are the syndromes of $1, 8, 9, 10$ ($x_2 = x_1$).

c) Case $\mathbf{n = 45, S_C = \{1, 21\}}$

This case is a covered by Theorem 6.2.5 for $l = 2$ and $s = 23$, which gives the following general error locator polynomial

$$z^2 + x_1 z + \left(\frac{x_2 x_4 + x_5}{x_3}\right)^{23}, \tag{6.17}$$

where x_1, x_2, x_3, x_4, x_5 are the syndromes of $1, 2, 19, 21, 23$. Note that the inverse of 2 modulo 45 is 23.

d) Case $\mathbf{n = 51, S_C = \{1, 9\}}$

This is a special case of those covered by Corollary 6.2.8 for $i = 0$ and $j = 4$ which gives the following general error locator polynomial

$$z^2 + x_1 z + \left(\frac{x_1^{18} + x_4}{x_1(x_1^{15} + x_3)}\right), \tag{6.18}$$

where x_1, x_3, x_4 are the syndromes of $1, 15, 18$.

e) Case $\mathbf{n = 51, S_C = \{0, 1, 5\}}$

For this code we get the general error locator polynomial as follows.
Thanks to Theorem 6.2.7 with $i = 0$ and $j = 3$, we have that $x_1(x_1^7 + x_3)b^* = (x_1^{10} + x_4)$ where x_1, x_3, x_4 are the syndromes of $1, 7, 10$ respectively. Now, for some locations (z_1, z_2) of weight

2, $(x_1^7 + x_3)$ becomes zero. However, when $(x_1^7 + x_3) = 0$, it is easy to get the value of b^*. Indeed,

$$x_1^7 = (z_1 + z_2)^7 = z_1^7 + z_2^7 + (z_1 z_2)\left((z_1 + z_2)^5 + (z_1 z_2)^2(z_1 + z_2)\right) \tag{6.19}$$

So, when $(x_1^7 + x_3) = 0$ and $\mu = 2$, we obtain that $\left((z_1 + z_2)^5 + b^{*2}(z_1 + z_2)\right) = 0$, which gives $b^* = x_1^2$.

In conclusion, if $(x_1^7 + x_3)$ is nonzero, then $b^* = (x_1^{10} + x_4)/(x_1(x_1^7 + x_3))$, otherwise, $b^* = x_1^2$.

To unify the two representations, we proceed as follows. By Lemma 6.2.9 we have that

$$(x_1^7 + x_3) = 0 \quad \text{if and only if} \quad x_1^{51} = 1. \tag{6.20}$$

Since $(x_1^{255} + 1) = (x_1^{51} + 1)F(x_1^{51})$ with $F(y) = y^4 + y^3 + y^2 + y + 1$, from (6.20) we get that a general error locator polynomial for C is

$$z^2 + x_1 z + \left(\frac{x_1^{10} + x_4}{x_1(x_1^7 + x_3)}\right) + x_1^2\left(\frac{F(x_1^{51})^2}{F(x_2^{51})}\right), \tag{6.21}$$

where x_1, x_2, x_3, x_4 are the syndromes of $1, 2, 7, 10$.

In Tables 6.1, 6.3 and 6.4 we list binary cyclic codes, up to equivalence and subcodes, with length less than 121 which are covered by Theorem 6.2.4, Theorem 6.2.6 and Corollary 6.2.8 respectively. While, Table 6.2 reports the codes with length $n < 105$ which are covered by Theorem 6.2.5.

Table 6.1: Binary cyclic codes with $t = 2$ and length < 121 covered by Theorem 6.2.4

$15, \{1, 3\}$	$17, \{1\}$	$21, \{1, 3\}$	$25, \{1\}$	$27, \{1, 9\}$	$31, \{1, 3\}$	$31, \{1, 5\}$
$35, \{1, 5\}$	$35, \{1, 3\}$	$45, \{1, 21\}$	$45, \{1, 3\}$	$45, \{1, 9\}$	$51, \{1, 3\}$	$51, \{1, 9\}$
$55, \{1\}$	$63, \{1, 3\}$	$65, \{1\}$	$73, \{1, 9\}$	$73, \{1, 3\}$	$73, \{1, 5\}$	$73, \{1, 17\}$
$75, \{1, 3\}$	$77, \{1, 33\}$	$81, \{1, 9\}$	$85, \{1, 3\}$	$85, \{1, 5\}$	$85, \{1, 9\}$	$91, \{1, 17\}$
$93, \{1, 3\}$	$93, \{1, 9\}$	$95, \{1\}$	$99, \{1, 33\}$	$99, \{1, 9\}$	$105, \{1, 3\}$	$115, \{1\}$
$117, \{1, 9\}$	$119, \{1, 17\}$	$119, \{1, 13\}$				

For the sake of space, we do not show the codes covered by Theorem 6.2.5 with length $105 \leq n < 121$. We observe that in each table we also report BCH codes.

6.3 Sparse locators for some classes of codes with $t = 3$

In this section we provide explicit sparse representations for some infinite classes of binary codes with error correction capability $t = 3$. We also consider all binary codes with $t = 3$ and $n < 63$, showing that they can be regrouped in few classes and we provide a general error

Table 6.2: Binary cyclic codes with $t = 2$ and length < 105 covered by Theorem 6.2.5

$9, \{0, 1\}$	$15, \{1, 3\}$	$15, \{0, 1, 7\}$	$17, \{0, 1\}$	$21, \{0, 1, 5\}$
$21, \{1, 3\}$	$25, \{1\}$	$27, \{1, 9\}$	$27, \{0, 1\}$	$31, \{0, 1, 15\}$
$31, \{1, 5\}$	$31, \{1, 3\}$	$33, \{0, 1\}$	$35, \{1, 3\}$	$35, \{1, 5\}$
$45, \{0, 1, 7\}$	$45, \{1, 3\}$	$45, \{1, 7, 15\}$	$45, \{1, 21\}$	$45, \{1, 9\}$
$51, \{1, 3\}$	$51, \{1, 9, 17\}$	$51, \{0, 1, 19\}$	$51, \{0, 1, 5, 11\}$	$55, \{1\}$
$63, \{1, 3\}$	$63, \{1, 5, 9\}$	$63, \{1, 27, 31\}$	$63, \{1, 9, 31\}$	$63, \{1, 11, 27\}$
$63, \{0, 1, 31\}$	$15, \{1, 21, 31\}$	$63, \{1, 5, 13, 21\}$	$63, \{1, 5, 11, 21\}$	$63, \{1, 23, 27, 31\}$
$63, \{1, 5, 9, 31\}$	$63, \{1, 7, 13, 27\}$	$63, \{1, 5, 11, 27\}$	$63, \{1, 23, 27, 31\}$	$63, \{1, 5, 11, 21, 31\}$
$65, \{1, 3\}$	$65, \{1, 7\}$	$65, \{0, 1\}$	$69, \{0, 1, 5\}$	$73, \{1, 3\}$
$73, \{1, 17\}$	$73, \{0, 1, 9\}$	$75, \{1, 3\}$	$75, \{0, 1, 7\}$	$77, \{1, 33\}$
$81, \{1, 27\}$	$81, \{0, 1\}$	$81, \{1, 9\}$	$85, \{1, 3\}$	$85, \{1, 29\}$
$85, \{1, 7, 13\}$	$85, \{1, 7, 9\}$	$85, \{1, 7, 21\}$	$85, \{1, 9, 17\}$	$85, \{1, 13, 37\}$
$85, \{1, 21, 37\}$	$85, \{0, 1, 21\}$	$87, \{0, 1, 5\}$	$91, \{0, 1, 17\}$	$91, \{1, , 9, 11, 13\}$
$93, \{1, 3\}$	$93, \{1, 5, 9\}$	$93, \{1, 5, 15\}$	$93, \{1, 5, 21\}$	$93, \{1, 5, 45\}$
$93, \{0, 1, 23\}$	$93, \{1, 11, 15\}$	$93, \{0, 1, 5, 11\}$	$95, \{1\}$	$99, \{1, 9\}$
$99, \{1, 33\}$	$99, \{0, 1\}$			

Table 6.3: Binary cyclic codes with $t = 2$ and length < 121 covered by Theorem 6.2.6

$17, \{1\}$	$25, \{1\}$	$31, \{1, 15\}$	$31, \{1, 3\}$	$43, \{1\}$	$55, \{1, 3\}$	$65, \{1\}$
$73, \{1, 9\}$	$85, \{1, 21\}$	$91, \{1, 3\}$	$95, \{1, 7\}$	$115, \{1, 7\}$	$119, \{1, 13\}$	

Table 6.4: Binary cyclic codes with $t = 2$ and length < 121 covered by Corollary 6.2.8

$15, \{1, 3\}$	$21, \{1, 3\}$	$25, \{1\}$	$31, \{1, 3\}$	$31, \{1, 5, 15\}$	$35, \{1, 3\}$	$45, \{1, 3\}$
$45, \{1, 9, 15\}$	$51, \{1, 3\}$	$51, \{1, 9\}$	$55, \{1, 3\}$	$63, \{1, 3\}$	$65, \{1, 3\}$	$73, \{1, 3\}$
$73, \{1, 5\}$	$73, \{1, 17\}$	$75, \{1, 3\}$	$85, \{1, 3\}$	$85, \{1, 9, 15\}$	$85, \{1, 9, 37\}$	$93, \{1, 3\}$
$93, \{1, 9, 15\}$	$95, \{1\}$	$105, \{1, 3\}$	$115, \{1\}$			

locator polynomial for all these codes. In [Che69] Chen produced a table of the minimum distances of binary cyclic codes of length at most 65. This table was extended to length at most 99 by Promhouse and Tavares [PT78].

The following theorem lists binary cyclic codes with $t = 3$ and $n < 63$ up to equivalence and subcodes that we obtain with MAGMA computer algebra system [BCP97].

Theorem 6.3.1. *Let C be an $[n, k, d]$ code with $d \in \{7, 8\}$ and $15 \leq n < 63$ (n odd). Then there are only three cases.*

1) Either C is one of the following:

$$n = 15, S_C = \{1, 3, 5\}, n = 21, S_C = \{1, 3, 5\}, S_C = \{1, 3, 7, 9\}, S_C = \{0, 1, 3, 7\};$$

$$n = 23, S_C = \{1\}, n = 31, S_C = \{1, 3, 5\}, S_C = \{0, 1, 7, 15\};$$

$$n = 35, S_C = \{1, 3, 5\}, S_C = \{1, 5, 7\}, n = 45, S_C = \{1, 3, 5\}, S_C = \{1, 5, 9, 15\};$$

$$n = 49, S_C = \{1, 3\}, n = 51, S_C = \{1, 3, 9\}, n = 55, S_C = \{0, 1\};$$

2) or C is a subcode of one of the codes of case 1);

3) or C is equivalent to one of the codes of the above cases.

Subcodes and equivalences are described in Table A.1 in the Appendix A. By Theorem 12 [OS07], we need to find a general error locator polynomial only for the codes in 1). For our purposes, it is convenient to regroup the codes as showed in the following theorem.

Theorem 6.3.2. *Let C be an $[n, k, d]$ code with $d \in \{7, 8\}$ and $15 \leq n < 63$ (n odd). Then there are six cases*

1) either C is a BCH code, i.e. $S_C = \{1, 3, 5\}$,

2) or C admits a defining set containing $\{1, i, i+1, i+2, i+3, i+4\}$ where i and $i+2$ are not zero modulo n,

3) or C admits a defining set containing $\{1, 3, 2^i + 2^j, 2^j - 2^i, 2^j - 2^{i+1}\}$ with $i \geq 0$ and $j \geq i + 2$,

4) or C admits a defining set containing $\{1, 3, 9\}$ and $(n, 3) = 1$,

5) or C is one of the following:

- $n = 21$, $S_C = \{0, 1, 3, 7\}$;
- $n = 51$, $S_C = \{1, 3, 9\}$;
- $n = 55$, $S_C = \{0, 1\}$.

6) or C is a subcode of one of the codes of the above cases,

7) or C is equivalent to one of the codes of the above cases.

Proof. It is enough to inspect Case 1) of Theorem 6.3.1 □

Corollary 6.3.3. *Let C be a code with length $n < 63$ and distance $d \in \{7, 8\}$. Then C is equivalent to a code D s.t $1 \in S_D$.*

Proof. It is an immediate consequence of Theorem 6.3.1. □

Let C be a code with $t = 3$, $\mathbf{s}$ a correctable syndrome and $\bar{z}_1$, $\bar{z}_2$, $\bar{z}_3$ the error locations. Then $\mathcal{L}(X, z) = z^3 + az^2 + bz + c$, where $a, b, c \in \mathbb{F}_2[X]$, and $a(\mathbf{s}) = \bar{z}_1 + \bar{z}_2 + \bar{z}_3$, $b(\mathbf{s}) = \bar{z}_1\bar{z}_2 + \bar{z}_1\bar{z}_3 + \bar{z}_2\bar{z}_3$, $c(\mathbf{s}) = \bar{z}_1\bar{z}_2\bar{z}_3$. Moreover, there are three errors if and only if $c(\mathbf{s}) \neq 0$, there are two errors if and only if $c(\mathbf{s}) = 0$ and $b(\mathbf{s}) \neq 0$, and there is one error if and only if $c(\mathbf{s}) = b(\mathbf{s}) = 0$ and $a(\mathbf{s}) \neq 0$. Note that from the previous corollary any code with $t = 3$ and

$n < 63$ is equivalent to a code with 1 in the defining set. This means that for all our codes the general error locator polynomial is of the form

$$\mathcal{L}(X, z) = z^3 + x_1 z^2 + bz + c,$$

where x_1 is the syndrome corresponding to $1 \in S_C$. So we are left with finding the coefficients b and c. Of course, b in the $t = 3$ case should not be confused with b in the case of $t = 2$ case. Also, when $3 \in S_C$, actually we need to find only one of the two coefficients because in this case by Newton's identities we get $c = x_1^3 + x_3 + x_1 b$, which involves only known syndromes, so from one coefficient we can easily obtain the other. In the following, $\Sigma_{l,m}$ will denote all the six terms of the type $z_i^l z_j^m$, $i, j \in \{1, 2, 3\}$, and $\Sigma_{l,m,r}$ denotes all the six terms of the type $z_i^l z_j^m z_k^r$, $i, j, k \in \{1, 2, 3\}$.

Let us consider the codes in 1) of Theorem 6.3.2. We have the following well-known result.

Theorem 6.3.4. *Let C be a BCH code with $t = 3$. Then $\mathcal{L}(X, z) = z^3 + x_1 z^2 + bz + c$ with*

$$b = \frac{(x_1^2 x_3 + x_5)}{(x_1^3 + x_3)}, \quad c = \frac{(x_1^3 x_3 + x_1^6 + x_3^2 + x_1 x_5)}{(x_1^3 + x_3)}$$

Proof. It enough to apply Newton's identities. □

The next theorem provides a general error locator polynomial for codes in 2) of Th. 6.3.2.

Theorem 6.3.5. *Let C be a code with $t = 3$ and S_C containing $\{1, i, i+1, i+2, i+3, i+4\}$ where i and $i+2$ are not zero modulo n. Then $\mathcal{L}(X, z) = z^3 + x_1 z^2 + bz + c$ with*

$$b = \frac{x_i U + x_{i+1} V}{W}, \quad c = \frac{x_{i+1} U + x_{i+2} V}{W}$$

where $U = x_{i+4} + x_1 x_{i+3}$, $V = x_{i+3} + x_1 x_{i+2}$ and $W = x_{i+1}^2 + x_i x_{i+2}$.

Proof. Let us suppose that three errors occur, that is, $\mathbf{e}$ has weight three, and let $\mathbf{s}$ be its syndrome vector. It is a simple computation to show, using the Newton's identities

$$\begin{cases} x_{i+4} = x_1 x_{i+3} + b x_{i+2} + c x_{i+1} \\ x_{i+3} = x_1 x_{i+2} + b x_{i+1} + c x_i \end{cases}$$

that $b = \frac{x_i U + x_{i+1} V}{W}$, and $c = \frac{x_{i+1} U + x_{i+2} V}{W}$, where $W = x_{i+1}^2 + x_i x_{i+2} = \Sigma_{i,i+2}$ which cannot be zero because i and $i+2$ are not zero modulo n. Then, when $\mu = 3$, $\mathcal{L}(\mathbf{s}, z)$ is the error locator polynomial for C.

Let us show that it is actually a general error locator polynomial for C. We have that

$$x_{i+1}U + x_{i+2}V = (z_1^{i+1} + z_2^{i+1} + z_3^{i+1}) \left(z_1^{i+4} + z_2^{i+4} + z_3^{i+4} + (z_1 + z_2 + z_3)(z_1^{i+3} + z_2^{i+3} + z_3^{i+3})\right)$$
$$+ (z_1^{i+2} + z_2^{i+2} + z_3^{i+2}) \left(z_1^{i+3} + z_2^{i+3} + z_3^{i+3} + (z_1 + z_2 + z_3)(z_1^{i+2} + z_2^{i+2} + z_3^{i+2})\right) = \Sigma_{1,i+1,i+3},$$

and

$$x_iU + x_{i+1}V = (z_1^i + z_2^i + z_3^i)\left(z_1^{i+4} + z_2^{i+4} + z_3^{i+4} + (z_1+z_2+z_3)(z_1^{i+3} + z_2^{i+3} + z_3^{i+3})\right) + (z_1^{i+1} + z_2^{i+1} + z_3^{i+1})\left(z_1^{i+3} + z_2^{i+3} + z_3^{i+3} + (z_1+z_2+z_3)(z_1^{i+2} + z_2^{i+2} + z_3^{i+2})\right) = = \Sigma_{1,i,i+3} + \Sigma_{1,i+1,i+2} + \Sigma_{i+1,i+3},$$

Let us suppose that $\mu = 2$. In this case, $W = z_1^i z_2^{i+2} + z_1^{i+2} z_2^i$, which is again different from zero. Furthermore, $x_{i+1}U + x_{i+2}V = \Sigma_{1,i+1,i+3}$ is zero because $\mu = 2$. Finally, $x_iU + x_{i+1}V$ is different from zero because $x_iU + x_{i+1}V = \Sigma_{1,i,i+3} + \Sigma_{1,i+1,i+2} + \Sigma_{i+1,i+3}$ and $\Sigma_{i+1,i+3}$ cannot be zero. When $\mu = 1$, $W = z_1^i z_2^{i+2} + z_1^{i+2} z_2^i = 0$ and $x_iU + x_{i+1}V = \Sigma_{i+1,i+3} = 0$. □

To obtain a general error locator polynomial for codes in 3) of Theorem 6.3.2, we need the following lemma.

Lemma 6.3.6. *Let $\sigma_k = \sum_{1\le i_1<\cdots<i_k\le 3} z_{i_1}\cdots z_{i_k}$ be the kth elementary symmetric polynomial in the variables z_1, z_2, z_3 over $\mathbb{F}_2$, where $k \in \{1,2,3\}$, and let $x_h = \sum_{l=1}^{3} z_l^h \in \mathbb{F}_2[z_1,z_2,z_3]$ be the power sum polynomial of degree h, with $h \ge 0$. Then, for $i \ge 0$ and $j \ge i+2$,*

$$x_1^{2^i+2^j} + x_{2^i+2^j} = \sigma_2^{2^i} x_{2^j-2^i} + \sigma_3^{2^i} x_{2^j-2^{i+1}}$$

Proof. $x_1^{2^i+2^j} = (z_1+z_2+z_3)^{2^i+2^j} = (z_1+z_2+z_3)^{2^i}(z_1+z_2+z_3)^{2^j} = x_{2^i+2^j} + \Sigma_{2^i,2^j}$. On the other hand, $\sigma_2^{2^i} x_{2^j-2^i} = (z_1z_2 + z_1z_3 + z_2z_3)^{2^i}(z_1^{2^j-2^i} + z_2^{2^j-2^i} + z_3^{2^j-2^i}) = \Sigma_{2^i,2^j} + \Sigma_{2^i,2^i,2^j-2^i}$ and $\sigma_3^{2^i} x_{2^j-2^{i+1}} = (z_1z_2z_3)^{2^i}(z_1^{2^j-2^{i+1}} + z_2^{2^j-2^{i+1}} + z_3^{2^j-2^{i+1}}) = \Sigma_{2^i,2^i,2^j-2^i}$. So $x_1^{2^i+2^j} = x_{2^i+2^j} + \sigma_2^{2^i} x_{2^j-2^i} + \sigma_3^{2^i} x_{2^j-2^{i+1}}$. □

Theorem 6.3.7. *Let C be a code with $t = 3$ and S_C containing $\{1, 3, 2^i+2^j, 2^j-2^i, 2^j-2^{i+1}\}$ with $i \ge 0$ and $j \ge i+2$. Then $\mathcal{L}(X,z) = z^3 + x_1z^2 + bz + c$ with*

$$b = \left(\frac{x_{2^j-2^{i+1}}U + V}{W}\right)^{(2^i)^+}, \quad c = \left(\frac{x_{2^j-2^i}U + x_1^{2^i}V}{W}\right)^{(2^i)^+}$$

where $U = (x_1^3 + x_3)^{2^i}$, $V = x_1^{2^i+2^j} + x_{2^i+2^j}$, $W = x_{2^j-2^i} + x_1^{2^i} x_{2^j-2^{i+1}}$ and $(2^i)^+$ is the inverse of 2^i modulo n.

Proof. Since the syndrome x_1 is a known syndrome, that is, $1 \in S_C$, we have that $a = x_1$. From the Newton identity $c = x_1^3 + x_3 + x_1b$ we get that

$$c^{2^i} = x_1^{3\times 2^i} + x_3^{2^i} + x_1^{2^i} b^{2^i} \tag{6.22}$$

On the other hand, by the previous lemma, we have that

$$x_1^{2^i+2^j} + x_{2^i+2^j} = b^{2^i} x_{2^j-2^i} + c^{2^i} x_{2^j-2^{i+1}} \tag{6.23}$$

Taking into account (6.22) and (6.23), a few computations lead to the equalities $b^{2^i} = \left(\frac{x_{2^j-2^{i+1}}U+V}{W}\right)$ and $c^{2^i} = \left(\frac{x_{2^j-2^i}U+x_1^{2^i}V}{W}\right)$. Suppose that $\mu = 3$. Then $W = (z_1^{2^j-2^i} + z_2^{2^j-2^i} + z_3^{2^j-2^i}) + (z_1^{2^i} + z_2^{2^i} + z_3^{2^i})(z_1^{2^j-2^{i+1}} + z_2^{2^j-2^{i+1}} + z_3^{2^j-2^{i+1}}) = \Sigma_{2^i,2^j-2^{i+1}}$. Since j is an integer, it is not possible that $2^i = 2^j - 2^{i+1}$, then W is different from zero. Also $x_{2^j-2^i}U + x_1^{2^i}V = \Sigma_{2^i,2^{i+1},2^j-2^i} + \Sigma_{2^i,2^i,2^j}$ and $x_{2^j-2^{i+1}}U + V = \Sigma_{2^i,2^{i+1},2^j-2^{i+1}} + \Sigma_{2^{i+1},2^j-2^i}$. From the previous computations we get that if $\mu = 2$ then $W \neq 0$, $x_{2^j-2^i}U + x_1^{2^i}V = 0$, and $x_{2^j-2^{i+1}}U + V \neq 0$. The last equality is because $\Sigma_{2^{i+1},2^j-2^i} \neq 0$. Furthermore, if $\mu = 1$, then $x_{2^j-2^{i+1}}U + V = 0$. □

Finally, let us consider the codes in $4)$ of Theorem 6.3.2. In [Eli87] Elia presents an algebraic decoding for the $(23, 12, 7)$ Golay code providing the error locator polynomials for μ errors, for μ from one to three. In [Lee11] Lee proves that the error locator polynomial $L^{(3)}$ corresponding to three errors is actually a weak error locator polynomial for this code. Notice that $L^{(3)}$ is a weak error locator polynomial for all cyclic codes C with $t = 3$, S_C containing $\{1, 3, 9\}$ and $(n, 3) = 1$. Next theorem proves that one can obtain a general error locator polynomial for these codes by slightly modifying $L^{(3)}$.

Theorem 6.3.8. *Let C be a code with $t = 3$ and S_C containing $\{1, 3, 9\}$ with $(n, 3) = 1$. Then $\mathcal{L}(X, z) = z^3 + x_1z^2 + bz + c$ with*

$$b = (x_1^2 + D^{l^*})h, \quad c = (x_3 + x_1D^{l^*})h,$$

where $D = \left(\frac{x_9+x_1^9}{x_3+x_1^3}\right) + (x_1^3 + x_3)^2$, $h = \frac{(x_1^3+x_3)}{(x_1x_2+x_3)}$, $l = 3$ and l^* is the inverse of l modulo $2^m - 1$ with $\mathbb{F}_{2^m}$ the splitting field of $x^n - 1$ over $\mathbb{F}_2$.

Proof. Since $1 \in S_C$, we have that $a = x_1$. From the following Newton identities

$$\begin{cases} x_9 = x_1x_8 + bx_7 + cx_6 \\ x_7 = x_1x_6 + bx_5 + cx_4 \\ x_5 = x_1x_4 + bx_3 + cx_2 \\ x_3 = x_1x_2 + bx_1 + c \end{cases}$$

using the equalities $x_6 = x_3^2$, and $x_{2^i} = x_1^{2^i}$ for $i \geq 0$, we get

$$\left(\frac{x_9 + x_1^9}{x_3 + x_1^3}\right) + (x_1^3 + x_3)^2 = (b + x_1^2)^3 \tag{6.24}$$

So $b = x_1^2 + D^{l^*}$. From $x_3 = x_1x_2 + bx_1 + c$, we find $c = x_3 + x_1D^{l^*}$. Let us prove that $\mathcal{L}$ is a general error locator polynomial. By Lemma 1 and Lemma 2 in [Lee11], it is enough to note that when there are less than two errors $h = 0$, while when there are two or three errors $h = 1$. □

Table 6.5: Binary cyclic codes with $t = 3$ and length < 121 covered by Theorem 6.3.5

$15, \{1, 3, 5\}$	$21, \{1, 3, 5\}$	$21, \{1, 5, 9\}$	$23, \{0, 1\}$	$31, \{0, 1, 7, 15\}$
$31, \{1, 3, 5\}$	$35, \{1, 3, 5\}$	$35, \{1, 5, 7\}$	$45, \{1, 3, 5\}$	$49, \{1, 3\}$
$63, \{1, 3, 5\}$	$63, \{1, 3, 11, 23, 27, 31\}$	$63, \{1, 5, 9, 13, 21\}$	$63, \{1, 3, 11, 13, 23\}$	$63, \{1, 5, 11, 13, 15\}$
$63, \{1, 15, 23, 31\}$	$63, \{1, 5, 13, 15, 21\}$	$63, \{0, 1, 15, 31\}$	$63, \{1, 5, 9, 13, 15\}$	$63, \{1, 11, 13, 15, 23, 27\}$
$69, \{1, 3, 23\}$	$69, \{0, 1, 3\}$	$75, \{1, 3, 5\}$	$75, \{1, 3, 25\}$	$77, \{1, 3\}$
$77, \{1, 7, 33\}$	$85, \{1, 3, 5\}$	$85, \{1, 7, 13, 15, 17\}$	$85, \{1, 15, 29, 37\}$	$85, \{0, 1, 21, 37\}$
$89, \{0, 1, 3\}$	$89, \{0, 1, 11\}$	$91, \{1, 3\}$	$91, \{1, 9, 19\}$	$91, \{1, 7, 9, 11, 13\}$
$93, \{1, 5, 17, 33\}$	$93, \{1, 7, 9, 17\}$	$93, \{1, 15, 17, 31, 33\}$	$93, \{1, 11, 23, 45\}$	$93, \{1, 17, 23, 31, 33\}$
$93, \{1, 9, 17, 33\}$	$93, \{1, 3, 5\}$	$105, \{1, 3, 5\}$	$105, \{1, 3, , 13, 25\}$	$105, \{1, 5, 9, 17\}$
$105, \{1, 9, 13, 25\}$	$105, \{1, 5, 7, 9, 11\}$	$105, \{1, 3, 9, 17, 25\}$	$105, \{1, 3, 17, 21, 25\}$	$105, \{1, 3, 11, 17, 45\}$
$105, \{1, 5, 9, 49\}$	$105, \{1, 3, 17, 25, 49\}$	$105, \{1, 9, 11, 13, 15, 17\}$	$105, \{1, 9, 13, 45, 49\}$	$105, \{1, 9, 17, 25, 49\}$
$105, \{1, 9, 11, 13, 45\}$	$105, \{0, 1, 9, 13\}$	$105, \{1, 3, 17, 35\}$	$113, \{0, 1\}$	$115, \{1, 23, 25\}$
$117, \{0, 1, 3\}$	$117, \{0, 1, 21, 29\}$	$119, \{1, 3\}$	$119, \{1, 7, 17\}$	$119, \{1, 11, 13\}$

Table 6.6: Binary cyclic codes with $t = 3$ and length < 121 covered by Theorem 6.3.7

$15, \{1, 3, 5\}$	$21, \{1, 3, 5\}$	$21, \{1, 3, 7, 9\}$	$31, \{1, 3, 5\}$	$35, \{1, 3, 5\}$	$45, \{1, 3, 5\}$	$49, \{1, 3\}$
$63, \{1, 3, 5\}$	$75, \{1, 3, 5\}$	$77, \{1, 3\}$	$85, \{1, 3, 5\}$	$91, \{1, 3\}$	$93, \{1, 3, 15, 31, 33\}$	$93, \{1, 3, 7, 9\}$
$93, \{1, 3, 5\}$	$105, \{1, 3, 5\}$	$117, \{1, 3, 7\}$	$119, \{1, 3\}$			

In Tables 6.5, 6.6 we list binary cyclic codes, up to equivalence and subcodes, with length less than 121 which are covered by Theorem 6.3.5 and Theorem 6.3.7 respectively. We observe that in each table we also report BCH codes.

Table 6.7 shows a general error locator polynomial for each code in Case 1) of Theorem 6.3.1 with $n < 55$. Since the codes in Cases 2) and 3) of Theorem 6.3.1 are equivalent or subcodes of the codes in Case 1), so (Theorem 3.3.26) their general error locator polynomial is the same or can be easily deduced from one of the general error locator polynomial in the table.

In Table 6.7 the codes are grouped according to increasing lengths and are specified with defining sets containing only primary syndromes. For each of these codes, the coefficients b and c of the general error locator polynomial is reported respectively in the second column and in the third column; The value in the fourth column explains which point of Theorem 6.3.2 has been used to describe the corresponding code family. In all cases except case 4 and for the codes with length $n = 49$ and $n = 51$, b and c are expressed in terms of primary syndromes: if the defining set in the last column is $S_C = \{i_1, i_2, \ldots, i_j\}$ with $i_1 < i_2 < \cdots < i_j$, then x_k denotes the syndrome corresponding to i_k, for $k = 1, 2, \ldots, j$. When 0 belongs to the defining set, it will be treated as if it were an n, with n the length of the code. For instance, for the code with length $n = 21$ and defining set $\{0, 1, 3, 7\}$ the syndrome corresponding to 0 is x_4. Codes described by the point 4 of Theorem 6.3.2 maintain the notation of Proposition

Table 6.7: Binary cyclic code with $t = 3$ and $n < 55$

n	b	c	Case	Codes
15	$\frac{(x_1^3x_2+x_1^6+x_2^2+x_1x_3)}{(x_1^3+x_2)}$	$\frac{(x_1^2x_2+x_3)}{(x_1^3+x_2)}$	1	$\{1,3,5\}$
21	$\frac{(x_1^3x_2+x_1^6+x_2^2+x_1x_3)}{(x_1^3+x_2)}$	$\frac{(x_1^2x_2+x_3)}{(x_1^3+x_2)}$	1	$\{1,3,5\}$
	$\frac{x_2^2(x_1^3+x_2)+(x_1^9+x_4)}{x_3+x_1x_2^2}$	$\frac{x_3(x_1^3+x_2)+x_1(x_1^9+x_4)}{x_3+x_1x_2^2}$	3	$\{1,3,7,9\}$
	$x_4x_1^2 + x_3^3x_1^2 + x_3^2x_2^3 + x_3^2x_2^2x_1^3 + x_3^2x_1^9+x_3x_2^3x_1^{28}+x_3x_2^2x_1^{10}+x_3x_2x_1^{13}+ x_3x_1^{37}+x_2^7x_1^{44}+x_2^7x_1^{23}+x_2^6x_1^{47}+x_2^6x_1^5+ x_2^5x_1^{50} + x_2^4x_1^{53} + x_2^4x_1^{32} + x_2^3x_1^{56} + x_2^3x_1^{35} + x_2^2x_1^{59} + x_2^2x_1^{38} + x_2x_1^{41} + x_2x_1^{20} + x_1^{23} + x_1^2$	$x_1^3 + x_2 + x_1b$	5	$\{0,1,3,7\}$
23	$\left(x_1^2 + \left(\frac{x_9 + x_1^9}{x_3 + x_1^3} + (x_1^3 + x_3)^2\right)^{1365}\right) \cdot \cdot \frac{(x_1^3 + x_3)}{(x_1x_2 + x_3)}$	$(x_1^3 + x_3 + bx_1)\frac{(x_1^3+x_3)}{(x_1x_2+x_3)}$	4	$\{1\}$
31	$\frac{(x_1^3x_2+x_1^6+x_2^2+x_1x_3)}{(x_1^3+x_2)}$	$\frac{(x_1^2x_2+x_3)}{(x_1^3+x_2)}$	1	$\{1,3,5\}$
	$\frac{x_3^8(x_4+x_1x_3^2)+x_2^4(x_3^2+x_1x_3^4)}{(x_3^{12}+x_2^8)}$	$\frac{x_2^4(x_4+x_1x_3^2)+x_3^4(x_3^2+x_1x_3^4)}{(x_3^{12}+x_2^8)}$	2	$\{0,1,7,15\}$
35	$\frac{(x_1^3x_2+x_1^6+x_2^2+x_1x_3)}{(x_1^3+x_2)}$	$\frac{(x_1^2x_2+x_3)}{(x_1^3+x_2)}$	1	$\{1,3,5\}$
	$\frac{x_3(x_1^{256}+x_1x_2^2)+x_1^8(x_2^2+x1^{1025})}{x_1^{16}+x_3x_1^{1024}}$	$\frac{x_1^8(x_1^{256}+x_1x_2^2)+x_1^{1024}(x_2^2+x1^{1025})}{x_1^{16}+x_3x_1^{1024}}$	2	$\{1,5,7\}$
45	$\frac{(x_1^3x_2+x_1^6+x_2^2+x_1x_3)}{(x_1^3+x_2)}$	$\frac{(x_1^2x_2+x_3)}{(x_1^3+x_2)}$	1	$\{1,3,5\}$,
	$\frac{x_4(x_1x_3^2+x_1^{64})+x_1^{16}(x_3^2+x_1^{513})}{x_1^{512}x_4+x_1^{32}}$	$\frac{x_1^{16}(x_1x_3^2+x_1^{64})+x_1^{512}(x_3^2+x_1^{513})}{x_1^{512}x_4+x_1^{32}}$	2	$\{1,5,9,15\}$
49	$\frac{(x_1^3x_2+x_1^6+x_2^2+x_1x_3)}{(x_1^3+x_2)}$	$\frac{(x_1^2x_2+x_3)}{(x_1^3+x_2)}$	1	$\{1,3\}$
51	$x_1^2 + (x_1^3 + x_2)(\frac{x_3^2+x_5x_3}{q_1x_1} + (\frac{x_3+x_2^3}{x_5^4+x_2^3} + 1)(\frac{x_1^2}{x_1^3+x_2} + \frac{x_4+x_2^4x_1}{q_2}))$ $q_1 = (x_3x_1^9 + x_3x_2x_1^6 + x_2^3x_1^9 + x_3^2 + x_3x_2^2x_1^3 + x_2^4x_1^6 + x_3x_2^3 + x_5x_1^3 + x_5x_2 + x_2^6)$ $q_2 = (x_1^{16} + x_2^4x_1^4 + x_4x_2 + x_2^5x_1)$	$x_1^3 + x_2 + x_1b$	5	$\{1,3,9\}$

6.3.8, so x_i denotes the syndrome corresponding to i. In the case of the code with $n = 49$, x_1, x_2, x_3 denote the syndromes corresponding to $1, 3, 5$ respectively, while for the codes with length 51, x_1, x_2, x_3, x_4, x_5 denote the syndromes corresponding to $1, 3, 9, 13, 15$ respectively. The coefficient a of the general error locator polynomial is not reported in Table 6.7 because any code in Case 1) of Theorem 6.3.1 has 1 in its defining set, so in all cases $a = x_1$. A general error locator for the codes with $t = 3$ and $n = 55$ is showed in Table A.2 in the Appendix A.

6.4 On the complexity of decoding cyclic codes

In this section we estimate the complexity of the decoding approach based on general error locator polynomials presented in Section 3.3.2 for any cyclic code, along with a comparison with similar approaches for the case where the generator polynomial of the cyclic code is irreducible.

6.4.1 Complexity of the proposed decoding approach

Definition 6.4.1. Let $\mathbb{K}$ be any field and let f be any (possibly multivariate) polynomial with coefficients in $\mathbb{K}$, that is, $f \in \mathbb{K}[\mathtt{a}_1, \ldots, \mathtt{a}_N]$ for a variable set $A = \{\mathtt{a}_1, \ldots, \mathtt{a}_N\}$. We will denote by $|f|$ the number of terms (monomials) of f.

Definition 6.4.2. Let $A = \{\mathtt{a}_1, \ldots, \mathtt{a}_N\}$ and $B = \{\mathtt{b}_1, \ldots, \mathtt{b}_M\}$ be two variable sets. Let $\mathbb{K}$ be a field and let $\mathcal{F}$ be a rational function in $\mathbb{K}(A)$. Let $F \in \mathbb{K}[B]$, $f_1, \ldots, f_M \in \mathbb{K}[A]$ and $g_1, \ldots, g_M \in \mathbb{K}[A]$. We say that the triple $(F, \{f_1 \ldots, f_M\}, \{g_1 \ldots, g_M\})$ is a **rational representation** of $\mathcal{F}$ if

$$\mathcal{F} = F(f_1/g_1, \ldots, f_M/g_M)$$

. We say that the number

$$|F| + \sum_{i=1}^{M}(|f_i| - 1) + \sum_{j=1, g_j \notin \mathbb{K}}^{M} |g_j|$$

is the **density** of the rational representation $(F, \{f_1 \ldots, f_M\}, \{g_1 \ldots, g_M\})$.
Then, we define the **functional density** of $\mathcal{F}$, $||\mathcal{F}||$, as the minimum among the densities of all rational representations of $\mathcal{F}$.

With the notation of Definition 6.4.1 and 6.4.2, we have the following result, that shows their interlink and how natural Definition 6.4.2 is.

Theorem 6.4.3. *Let $A = \{\mathtt{a}_1, \ldots, \mathtt{a}_N\}$. If $\mathcal{F}$ is a polynomial in $\mathbb{K}[A]$, rather than a rational function in $\mathbb{K}(A)$, then*

$$||\mathcal{F}|| \leq |\mathcal{F}|\,.$$

Moreover, if $\mathcal{F} = \mathtt{a}_1 + \mathtt{a}_2$ then $||\mathcal{F}|| = |\mathcal{F}| = 2$.

Proof. Let $\mathcal{F} \in \mathbb{K}[A]$ and let $\rho = |\mathcal{F}|$. Then $\mathcal{F} = \sum_{i=1}^{\rho} h_i$, where any h_i is a monomial for $1 \leq i \leq \rho$.
Let us consider the following rational representation for $\mathcal{F}$

$$B = \{\mathtt{b}_1, \ldots, \mathtt{b}_\rho\}\,, \quad F = \sum_{i=1}^{\rho} \mathtt{b}_i\,, \quad f_i = h_i\,, \quad g_i = 1,\ 1 \leq i \leq \rho\,,$$

then the rational density of $(F, \{f_1, \ldots, f_\rho\}, \{g_1, \ldots, g_\rho\})$ is

$$|F| + \sum_{i=1}^{\rho}(|f_i| - 1) + \sum_{j=1, g_j \notin \mathbb{K}}^{\rho} |g_j| \quad = \rho + 0 + 0 = \rho\,,$$

which implies $||\mathcal{F}|| \leq \rho$, as claimed.

To prove the case $\mathcal{F} = \mathtt{a}_1 + \mathtt{a}_2$, we argument by contradiction assuming $||\mathcal{F}|| = 1$. Let us consider the rational representation of $\mathcal{F}$ providing

$$||\mathcal{F}|| \quad = \quad |F| + \sum_{i=1}^{M}(|f_i| - 1) + \sum_{j=1, g_j \notin \mathbb{K}}^{M} |g_j| \quad = \; 1\,.$$

Since $|F| \geq 1$, we must have $|F| = 1$, $\sum_{i=1}^{M}(|f_i| - 1) = 0$ and $\sum_{j=1, g_j \notin \mathbb{K}}^{M} |g_j| = 0$.
Therefore, $M = 1$, $|f_1| = 1$ and $g_1 = \nu \in \mathbb{K}$. From $M = 1$ and $|F| = 1$ we have $F = \lambda \mathtt{b}_1^{\mu}$ for $\lambda \in \mathbb{K}$ and $\mu \geq 1$, and so $\mathcal{F} = F(f_1/g_1) = (\frac{f_1}{\nu})^{\mu}$. Recalling that $\mathcal{F} = \mathtt{a}_1 + \mathtt{a}_2$, we finally have a contradiction

$$|f_1| = 1 \implies \Big|\Big(\frac{f_1}{\nu}\Big)^{\mu}\Big| = 1\,, \qquad \text{but} \quad |\mathcal{F}| = |\mathtt{a}_1 + \mathtt{a}_2| = 2\,.$$

□

For example, the locator $\mathcal{L} \in \mathbb{F}_2[z, x_1, x_2, x_3, x_4, x_5]$ for the case treated in Theorem 6.2.5 can be easily shown to have functional density $||\mathcal{L}|| \leq 6$, thanks to the following rational representation

$$\mathcal{L} = F(f_1/g_1, f_2/g_2, f_3/g_3),$$

where $F \in \mathbb{F}_2[\mathtt{b}_1, \mathtt{b}_2, \mathtt{b}_3]$, $f_1, f_2, f_3, g_1, g_2, g_3 \in \mathbb{F}_2[z, x_1, x_2, x_3, x_4, x_5]$ and

$$F = b_1^2 + b_1 b_2 + b_3^{l^+}, \quad f_1 = z, g_1 = 1,\ f_2 = x_1, g_2 = 1,\ f_3 = x_2 x_4 + x_5, g_3 = x_3\,.$$

Conjecture 6.4.4 (Sala, MEGA2005)**.**
Let $p \geq 2$ be a prime, $m \geq 1$ a positive integer and let $q = p^m$. There is an integer $\epsilon = \epsilon(q)$ such that for any cyclic code C over the field $\mathbb{F}_q$ with $n \geq q^4 - 1$, $\gcd(n, q) = 1$, $3 \leq d \leq n - 1$, C admits a general error locator polynomial $\mathcal{L}_c$ whose functional density is bounded by

$$||\mathcal{L}_c|| \leq n^{\epsilon}\,.$$

Moreover, for binary codes we have $\epsilon = 3$, that is, $\epsilon(2) = 3$.

Let C be a cyclic code over $\mathbb{F}_q$ of length n. Let d be its distance, t its error correction capability and $S_C = \{i_1, \ldots, i_r\}$ a defining set of C. Let $\mathcal{L}_C$ be a general error locator polynomial of C.

Definition 6.4.5. If $\mathcal{L}_C \in \mathbb{F}_2[x_1, \ldots, x_r]$, then we say that $\mathcal{L}_C$ is **sparse** if $||\mathcal{L}_C|| \leq n^3$.
If Conjecture 6.4.4 holds and $\mathcal{L}_C \in \mathbb{F}_q[x_1, \ldots, x_r]$, then we say that $\mathcal{L}_C$ is **sparse** if $||\mathcal{L}_C|| \leq n^{\epsilon}$.

The decoding procedure developed by Orsini and Sala in [OS07] consists of four steps:

1. Computation of the r syndromes $s_1, \ldots, s_r$ corresponding to the received vector;
2. Evaluation of $\mathcal{L}_C(x_1, \ldots, x_r, z)$ at $\mathbf{s} = (s_1, \ldots, s_r)$;
3. Computation of the roots of $\mathcal{L}_C(\mathbf{s}, z)$;
4. Computation of the error values $e_{l_1}, \ldots, e_{l_\mu}$

By analyzing the above decoding algorithm, we observe that the main computational cost is the evaluation of the polynomial $\mathcal{L}_C(x_1, \ldots, x_r, z)$ at $\mathbf{s}$, which reduces to the evaluation of its z-coefficients. Indeed, the computation of the r syndromes $s_1, \ldots, s_r$ and of the roots of $\mathcal{L}_C(\mathbf{s}, z)$ cost, respectively, $O(t\sqrt{n})$ and $\max(O(t\sqrt{n}), O(t\log(\log(t))\log(n)))$ ([SER11]), while the computation of the error values using Forney's algorithm costs $O(t^2)$ ([HV95]). Therefore, we can bound the total cost of steps 1, 3 and 4 with $O(n^2)$.
The following theorem is then clear and should be compared with the results in [BN90], which suggest that for linear codes an extension of Conjecture 6.4.4 is very unlikely to hold.

Theorem 6.4.6. *Let us consider all cyclic codes over the same field $\mathbb{F}_q$ with $\gcd(n, q) = 1$ and $d \geq 3$. If Conjecture 6.4.4 holds, they can be decoded in polynomial time in n, once a preprocessing has produced sparse general error locator polynomials.*

Proof. The only special situations not tackled by Conjecture 6.4.4 are the finite cases when $n < q^4 - 1$, which of course do not influence the asymptotic complexity, and the degenerate case when $d = n$, which can be decoded in polynomial time without using the general error locator algorithm. □

Although all reported experiments (at least in the binary case) confirm Conjecture 6.4.4, we are far from having a formal proof of it, therefore we pass to estimate the cost of the crucial step 2 starting from results claimed in this paper or found elsewhere in the literature.
To estimate the cost of evaluating the polynomial $\mathcal{L}_C$ (at the syndrome vector $\mathbf{s}$) we will mainly use Corollary 6.1.4, its consequences for the case $\lambda = n$ (which we can always choose), and the corresponding degree bound. We can neglect the cost of computing the values $\frac{x_h}{x_1^{i_h}}$ and consider polynomials in the new obvious variables. In [BES13], Ballico, Elia and Sala describe a method to evaluate a polynomial in $\mathbb{F}_q[x_1, \ldots, x_r]$ of degree δ with a complexity $O(\delta^{r/2})$. To estimate our δ, we observe that, by Corollary 6.1.4, we have a bound on the degree of each z-coefficient of $\mathcal{L}_C$ in any new variable and so its total degree is at most

$$\delta \quad \leq \quad \left((q^m - 1)(r - 1) + \frac{q^m - 1}{n} \right),$$

then, using the method in [BES13], the evaluation of (the z-coefficients of) $\mathcal{L}_c$ at $\mathbf{s}$ costs

$$O\left(t \left((q^m - 1)(r - 1) + \frac{q^m - 1}{n} \right)^{r/2} \right). \tag{6.25}$$

So, we get that the cost of the decoding approach we are proposing is given by

$$O\left(n^2 + t\left((q^m-1)(r-1) + \frac{q^m-1}{n}\right)^{r/2}\right). \tag{6.26}$$

We are going to show that there are infinite families of codes for which this approach is competitive with more straightforward methods (even for low values of t).

Let us fix the number of syndromes r, and let γ be an integer $\gamma \geq 1$. Let $\mathcal{C}^q_{r,\gamma}$ be the set of all codes over $\mathbb{F}_q$ with length n such that the splitting field of $x^n - 1$ over $\mathbb{F}_q$ is $q^m - 1 = O(n^\gamma)$ (and $\gcd(n, q) = 1$). For codes in $\mathcal{C}^q_{r,\gamma}$, the complexity (6.26) of this decoding depends on r and it is

$$r \geq 2, \quad O\left(tn^{\gamma r/2}\right), \qquad\qquad r = 1, \quad O(n^2 + tn^{\frac{\gamma-1}{2}}). \tag{6.27}$$

So, any family $\mathcal{C}^q_{r,\gamma}$ provides a class containing infinite codes which can be decoded in polynomial time, with infinite values of distance and length. Obviously, these classes extend widely the classes which are known to be decodable in polynomial time up to the *actual distance*.

Theorem 6.2.4,6.2.5,6.2.6, Corollary 6.2.8 and Theorem 6.3.5, 6.3.7, 6.3.8 show cases where the previous estimation can be drastically improved, at least for $t = 2$ and $t = 3$. Indeed, these theorems provide (infinite) classes of codes with $t = 2$ and $t = 3$ for which the evaluation of $\mathcal{L}_C$ costs $O(1)$, and so the decoding process costs $O(n^2)$. For $t = 2$ and $t = 3$ exhaustive searching method cost, respectively, $O(n^2)$ and $O(n^3)$. For $t = 2$ we match the best-known complexity and for $t = 3$ our method is better.

6.4.2 Comparison with other approaches

In the last years, several methods were proposed for decoding *binary* quadratic residue (QR) codes generated by irreducible polynomials. In [CL10], Chang and Lee propose three algebraic decoding algorithms based on Lagrange Interpolation Formula (LIF) for these codes. They introduce a variation for the general error locator polynomial, which we may call fixed-weight locator. A *fixed-weight locator* is a polynomial able to correct all errors of a fixed weight via the evaluation of the corresponding syndromes. They develop a method to obtain a representation of the primary unknown syndrome in terms of the primary known syndrome and a representation of the coefficients of both fixed-weight locator and general error locator polynomial for these codes. These polynomials are explicitly obtained for the $(17, 9, 5)$, $(23, 12, 7)$, $(41, 21, 9)$ QR codes. In Table 6.8 we treat these three codes one per column showing the number of terms relevant to the alternative representations. For each code, the second row deals with representation of the chosen primary unknown syndrome, while the last deal with two locators.

Note that, for all the three codes, the general error locator polynomials are sparse (even without using the rational representation) as forseen in Conjecture 6.4.4. In particular the $(41, 21, 9)$ code has error correction capability $t = 4$ and the number of terms of its locator is less than

Table 6.8: Number of terms of unknown syndrome, fixed-weight locator and general error locator

	$(17, 9, 5)$	$(23, 12, 7)$	$(41, 21, 9)$
Splitting field	$\mathbb{F}_{2^8}$	$\mathbb{F}_{2^{11}}$	$\mathbb{F}_{2^{20}}$
Unknown syndrome	5	17	1355
Fixed-weight locator	4	15	1270
General error locator	4	76	1380

$n^\epsilon = 41^3 = 68921$. Observe also that the evaluation of the locators of the $(23, 12, 7)$ code in Table 6.7 and in [CL10] cost approximately the same.

In [LCTC10], Chang et al. propose to decode *binary* cyclic codes generated by irreducible polynomials using, as in [CL10], an interpolation formula in order to get the general error locator polynomial but in a slightly different way. The general error locators they obtain satisfy at least one congruence relation, and they are explicitly found for the $(17, 9, 5)$ QR code, the $(23, 12, 7)$ Golay code, and one $(43, 29, 6)$ cyclic code. Table 6.9 shows the maximum number of terms for the coefficients of these three polynomials. Also in this case, the locators are *sparse*

Table 6.9: Maximum number of terms among the locator coefficients σ_i

	$(17, 9, 5)$	$(23, 12, 7)$	$(43, 29, 6)$
Splitting field	$\mathbb{F}_{2^8}$	$\mathbb{F}_{2^{11}}$	$\mathbb{F}_{2^{14}}$
General error locator	9	203	25

for the three codes.

In [LJM12], Lee et al. extend the method proposed by Chang and Lee in [CL10] for finding fixed-weight locators and general error locators for binary cyclic codes generated by irreducible polynomials to the case of *ternary cyclic codes* generated by irreducible polynomials. These polynomials are presented for two *ternary* cyclic codes, one $(11, 6, 5)$ code and one $(23, 12, 8)$ code. In Table 6.10 we report the maximum number of terms of the coefficients of the general error locator for these two codes.

To discuss the *sparsity* of these cases one would need to know $\epsilon(3)$ from Conjecture 6.4.4. Assuming an optimistic stance, let us compare their sparsity with $\epsilon(3) = 3$, that is, let us assume the polynomial exponent of the ternary codes to be the same as that of binary codes (reasonably $\epsilon(3) \geq \epsilon(2)$).

The first locator is definitely sparse, with $|\mathcal{L}| = 232 < 1331 = 11^3$. For the second locator we

Table 6.10: Maximum number of terms among the locator coefficients σ_i

	$(11, 6, 5)$	$(23, 12, 8)$
Splitting field	$\mathbb{F}_{3^5}$	$\mathbb{F}_{3^{11}}$
General error locator	232	15204

have $|\mathcal{L}| = 15204$ which compared to $n^3 = 23^3 = 12167$ show that the locator is not sparse (although the numbers are close) and indeed we believe much sparser locators exist for this code, still to be found.

In the same paper ([LJM12]) the authors give also an interesting upper bound on $|\mathcal{L}|$ which holds for any irreducible ternary cyclic code, as follows.

Proposition 6.4.7 ([LJM12])**.** *Let C be a ternary cyclic code of length n with defining set $S_C = \{1\}$, and error correction capability t. Each coefficient of a general error locator polynomial can be expressed as a polynomial in terms of the known syndrome x_1 and the number of terms of this polynomial is less than $\lfloor \frac{\sum_{\nu=1}^{t} 2^{\nu} \binom{n}{\nu}}{n} \rfloor$.*

Indeed, we can generalize their result to the following theorem holding over any finite field.

Theorem 6.4.8. *Let C be any cyclic code over $\mathbb{F}_q$ of length n with defining set $S_C = \{1\}$, $\gcd(n, q) = 1$ and error correction capability t. Each coefficient of a general error locator polynomial can be expressed as a polynomial in terms of the known syndrome x_1 and the number of terms of this polynomial is less than $\lfloor \frac{\sum_{\nu=1}^{t} (q-1)^{\nu} \binom{n}{\nu}}{n} \rfloor$.*

Proof. By considering Corollary 6.1.4 and the fact that to obtain any locator coefficient, one can use simply (univariate) Lagrange interpolation on the set of correctable syndromes, which are obviously $1 + \sum_{\nu=1}^{t} (q-1)^{\nu} \binom{n}{\nu}$. □

With q fixed, the codes covered by the previous theorem are actually the component of our families $\mathcal{C}_{1,\gamma}^{q}$ for $\gamma \geq 1$. Depending on the actual considered length we will have the correct determination of γ, since this value strongly depends on the size of the splitting field. By (6.26) case $r = 1$, the time complexity of the decoding method for codes in $\mathcal{C}_{1,\gamma}^{q}$ is

$$O\left(n^2 + tn^{(\gamma-1)/2}\right). \tag{6.28}$$

Using the estimation given by Proposition 6.4.7, the complexity of the same decoding approach for these codes is

$$O\left(n^2 + tn^{t-1}\right). \tag{6.29}$$

We observe that which of the two estimations is better depends on the particular values of t and γ.

Part III

Appendices

Some tables

Table A.1 report the binary cyclic codes with $t = 3$ and $n < 63$ grouped according to increasing lengths, and, within the same length according to Theorem 6.3.1, i.e. if two codes with the same length are equivalent or one is a subcode of the other,then they are in the same group.

Table A.1: Binary cyclic codes with $t = 3$ and $n < 63$

n	Codes
15	$\{\mathbf{1,3,5}\}$, $\{3,5,7\}$, $\{0,3,5,7\}$, $\{0,1,3,5\}$
21	$\{\mathbf{1,3,5}\}$, $\{1,5,9\}$, $\{1,3,5,9\}$
	$\{\mathbf{1,3,7,9}\}$, $\{3,5,7,9\}$, $\{0,3,5,7,9\}$, $\{0,1,3,7,9\}$
	$\{\mathbf{0,1,3,7}\}$, $\{0,5,7,9\}$, $\{0,1,3,7,9\}$, $\{0,3,5,7,9\}$
23	$\{\mathbf{1}\}$, $\{5\}$, $\{0,1\}$, $\{0,5\}$
31	$\{\mathbf{1,3,5}\}$, $\{1,5,7\}$, $\{3,5,15\}$, $\{3,11,15\}$, $\{0,1,5,7\}$, $\{0,3,5,15\}$, $\{0,1,3,5\}$, $\{1,3,11\}$, $\{1,7,11\}$, $\{0,1,7,11\}$, $\{0,1,3,11\}$, $\{5,7,15\}$, $\{0,5,7,15\}$, $\{7,11,15\}$, $\{0,3,11,15\}$
	$\{\mathbf{0,1,7,15}\}$, $\{0,1,3,15\}$, $\{0,3,7,11\}$, $\{0,5,11,15\}$, $\{0,1,5,11\}$, $\{0,3,5,7\}$
35	$\{\mathbf{1,3,5}\}$, $\{1,3,15\}$, $\{1,3,5,15\}$
	$\{\mathbf{1,5,7}\}$, $\{3,7,15\}$ $\{0,3,5,7,15\}$, $\{0,1,5,7,15\}$, $\{0,3,7,15\}$, $\{0,1,5,7\}$, $\{3,5,7,15\}$, $\{1,5,7,15\}$
45	$\{\mathbf{1,3,5}\}$, $\{1,5,21\}$, $\{5,7,21\}$, $\{3,5,7\}$, $\{0,1,3,5,9,21\}$, $\{3,5,7,9,15\}$, $\{1,3,5,9,21\}$, $\{1,5,9,21\}$, $\{1,5,15,21\}$, $\{0,1,5,9,21\}$, $\{0,1,5,15,21\}$, $\{0,3,5,7,9,15\}$, $\{1,3,5,9\}$, $\{0,3,5,7,15,21\}$, $\{0,1,3,5,9\}$, $\{3,5,7,15,21\}$, $\{0,3,5,7,21\}$, $\{1,5,9,15,21\}$, $\{3,5,7,21\}$, $\{0,3,5,7,9,15,21\}$, $\{0,1,3,5\}$, $\{5,7,9,21\}$, $\{1,3,5,9,15\}$, $\{0,5,7,9,21\}$, $\{0,1,5,9,15,21\}$, $\{0,1,3,5,9,15\}$, $\{0,1,3,5,15,21\}$, $\{3,5,7,9,15,21\}$, $\{1,3,5,15,21\}$, $\{3,5,7,15\}$, $\{0,3,5,7,15\}$,$\{0,1,3,5,9,15,21\}$, $\{5,7,15,21\}$, $\{1,3,5,9,15,21\}$,$\{0,5,7,15,21\}$, $\{0,3,5,7,9,21\}$, $\{0,1,3,5,21\}$, $\{1,3,5,15\}$, $\{3,5,7,9,21\}$, $\{5,7,9,15,21\}$, $\{3,5,7,9\}$, $\{0,1,3,5,15\}$, $\{1,3,5,21\}$, $\{0,5,7,9,15,21\}$, $\{0,1,5,21\}$, $\{0,3,5,7,9\}$, $\{0,3,5,7\}$, $\{5,7,21\}$, $\{0,5,7,21\}$
	$\{\mathbf{1,5,9,15}\}$, $\{5,7,9,15\}$, $\{3,5,7,9,15\}$, $\{0,3,5,7,9,15\}$, $\{1,5,9,15,21\}$, $\{0,3,5,7,9,15,21\}$, $\{1,3,5,9,15\}$, $\{0,1,5,9,15,21\}$,$\{0,5,7,9,15\}$, $\{0,1,3,5,9,15\}$, $\{3,5,7,9,15,21\}$, $\{0,1,3,5,9,15,21\}$, $\{1,3,5,9,15,21\}$, $\{0,5,7,9,15,21\}$, $\{0,1,5,9,15\}$
49	$\{\mathbf{1,3}\}$
51	$\{\mathbf{1,3,9}\}$, $\{3,9,11\}$, $\{3,9,19\}$, $\{3,5,9\}$ $\{1,3,9,17\}$, $\{0,1,3,9\}$, $\{3,5,9,17\}$, $\{0,3,5,9,17\}$, $\{3,9,11,17\}$, $\{0,3,9,11\}$, $\{3,9,17,19\}$, $\{0,3,9,11,17\}$, $\{0,3,9,17,19\}$, $\{0,3,9,19\}$, $\{0,1,3,9,17\}$, $\{0,3,5,9\}$
55	$\{\mathbf{0,1}\}$,$\{0,3\}$

For each group there is a code in bold, which is the one reported in Table 6.7, i.e. the code

for which we determined a general error locator polynomial and that can be used to obtain locators for all the codes of the group.

In Table A.2 we show the coefficients b and c of a general error locator polynomial for binary cyclic codes with $t = 3$ and $n = 55$. For the sake of conciseness, both b and c are represented in the form described in Theorem 6.1.3, where y_1 stands for x_1^{55}.

Table A.2: General error locator for cyclic codes with $t = 3$ and $n = 55$

b	$x_1^2 \cdot (y_1^{475}+y_1^{472}+y_1^{470}+y_1^{469}+y_1^{468}+y_1^{463}+y_1^{462}+y_1^{461}+y_1^{460}+y_1^{458}+y_1^{457}+y_1^{455}+y_1^{454}+y_1^{452}+y_1^{449}+y_1^{448}+y_1^{446}+y_1^{444}+$ $y_1^{443}+y_1^{440}+y_1^{436}+y_1^{434}+y_1^{427}+y_1^{426}+y_1^{425}+y_1^{424}+y_1^{417}+y_1^{416}+y_1^{413}+y_1^{410}+y_1^{408}+y_1^{405}+y_1^{403}+y_1^{402}+y_1^{401}+y_1^{399}+$ $y_1^{397}+y_1^{395}+y_1^{394}+y_1^{392}+y_1^{388}+y_1^{387}+y_1^{386}+y_1^{384}+y_1^{380}+y_1^{378}+y_1^{377}+y_1^{376}+y_1^{375}+y_1^{374}+y_1^{372}+y_1^{370}+y_1^{369}+y_1^{368}+$ $y_1^{364}+y_1^{363}+y_1^{361}+y_1^{360}+y_1^{359}+y_1^{358}+y_1^{357}+y_1^{355}+y_1^{350}+y_1^{347}+y_1^{345}+y_1^{343}+y_1^{340}+y_1^{338}+y_1^{336}+y_1^{334}+y_1^{330}+y_1^{329}+$ $y_1^{327}+y_1^{326}+y_1^{325}+y_1^{324}+y_1^{321}+y_1^{319}+y_1^{318}+y_1^{316}+y_1^{315}+y_1^{312}+y_1^{308}+y_1^{306}+y_1^{305}+y_1^{302}+y_1^{301}+y_1^{296}+y_1^{295}+y_1^{292}+$ $y_1^{290}+y_1^{289}+y_1^{285}+y_1^{284}+y_1^{278}+y_1^{277}+y_1^{276}+y_1^{275}+y_1^{274}+y_1^{273}+y_1^{272}+y_1^{271}+y_1^{265}+y_1^{261}+y_1^{260}+y_1^{256}+y_1^{255}+y_1^{250}+$ $y_1^{249}+y_1^{248}+y_1^{247}+y_1^{243}+y_1^{242}+y_1^{240}+y_1^{239}+y_1^{235}+y_1^{234}+y_1^{233}+y_1^{231}+y_1^{230}+y_1^{229}+y_1^{227}+y_1^{225}+y_1^{224}+y_1^{222}+y_1^{221}+$ $y_1^{217}+y_1^{215}+y_1^{213}+y_1^{212}+y_1^{210}+y_1^{209}+y_1^{207}+y_1^{205}+y_1^{203}+y_1^{202}+y_1^{201}+y_1^{200}+y_1^{199}+y_1^{197}+y_1^{195}+y_1^{189}+y_1^{187}+y_1^{183}+$ $y_1^{182}+y_1^{181}+y_1^{180}+y_1^{179}+y_1^{178}+y_1^{175}+y_1^{172}+y_1^{169}+y_1^{167}+y_1^{165}+y_1^{164}+y_1^{163}+y_1^{160}+y_1^{159}+y_1^{157}+y_1^{155}+y_1^{154}+y_1^{145}+$ $y_1^{141}+y_1^{137}+y_1^{133}+y_1^{130}+y_1^{129}+y_1^{128}+y_1^{125}+y_1^{123}+y_1^{122}+y_1^{121}+y_1^{117}+y_1^{115}+y_1^{114}+y_1^{113}+y_1^{112}+y_1^{111}+y_1^{110}+y_1^{109}+$ $y_1^{108}+y_1^{107}+y_1^{102}+y_1^{98}+y_1^{96}+y_1^{95}+y_1^{90}+y_1^{89}+y_1^{88}+y_1^{86}+y_1^{84}+y_1^{83}+y_1^{81}+y_1^{80}+y_1^{78}+y_1^{77}+y_1^{76}+y_1^{74}+y_1^{72}+y_1^{70}+y_1^{68}+$ $y_1^{67}+y_1^{65}+y_1^{63}+y_1^{62}+y_1^{61}+y_1^{55}+y_1^{54}+y_1^{53}+y_1^{52}+y_1^{51}+y_1^{50}+y_1^{49}+y_1^{47}+y_1^{46}+y_1^{45}+y_1^{43}+y_1^{42}+y_1^{40}+y_1^{38}+y_1^{36}+y_1^{35}+y_1^{33}+$ $y_1^{32}+y_1^{31}+y_1^{30}+y_1^{29}+y_1^{28}+y_1^{24}+y_1^{23}+y_1^{22}+y_1^{21}+y_1^{20}+y_1^{17}+y_1^{15}+y_1^{14}+y_1^{13}+y_1^{11}+y_1^{12}+y_1^9+y_1^7+y_1^6+y_1^4+y_1^3+y_1^2+y_1^1+1+$ $x_2 \cdot (y_1^{26}+y_1^{24}+y_1^{23}+y_1^{13}+y_1^{11}+y_1^{10}+y_1^8+y_1^7+y_1^6+y_1^3+y_1))$
c	$x_1^3 \cdot (y_1^{477}+y_1^{476}+y_1^{473}+y_1^{472}+y_1^{470}+y_1^{469}+y_1^{466}+y_1^{463}+y_1^{461}+y_1^{459}+y_1^{458}+y_1^{457}+y_1^{456}+y_1^{453}+y_1^{452}+y_1^{451}+y_1^{450}+y_1^{449}+$ $y_1^{448}+y_1^{447}+y_1^{446}+y_1^{443}+y_1^{441}+y_1^{440}+y_1^{439}+y_1^{438}+y_1^{436}+y_1^{433}+y_1^{431}+y_1^{428}+y_1^{422}+y_1^{420}+y_1^{419}+y_1^{414}+y_1^{413}+y_1^{410}+$ $y_1^{409}+y_1^{407}+y_1^{406}+y_1^{403}+y_1^{402}+y_1^{400}+y_1^{399}+y_1^{394}+y_1^{391}+y_1^{388}+y_1^{385}+y_1^{384}+y_1^{383}+y_1^{382}+y_1^{381}+y_1^{379}+y_1^{373}+y_1^{372}+y_1^{368}+$ $y_1^{367}+y_1^{366}+y_1^{363}+y_1^{362}+y_1^{359}+y_1^{358}+y_1^{357}+y_1^{356}+y_1^{354}+y_1^{353}+y_1^{350}+y_1^{349}+y_1^{348}+y_1^{347}+y_1^{344}+y_1^{342}+y_1^{341}+y_1^{340}+y_1^{339}+$ $y_1^{337}+y_1^{335}+y_1^{334}+y_1^{333}+y_1^{332}+y_1^{331}+y_1^{330}+y_1^{328}+y_1^{325}+y_1^{324}+y_1^{323}+y_1^{322}+y_1^{321}+y_1^{320}+y_1^{319}+y_1^{313}+y_1^{312}+y_1^{310}+y_1^{307}+$ $y_1^{305}+y_1^{304}+y_1^{303}+y_1^{302}+y_1^{300}+y_1^{295}+y_1^{294}+y_1^{293}+y_1^{292}+y_1^{289}+y_1^{287}+y_1^{286}+y_1^{283}+y_1^{282}+y_1^{280}+y_1^{279}+y_1^{276}+y_1^{274}+y_1^{272}+$ $y_1^{270}+y_1^{269}+y_1^{267}+y_1^{264}+y_1^{263}+y_1^{262}+y_1^{261}+y_1^{259}+y_1^{256}+y_1^{255}+y_1^{254}+y_1^{253}+y_1^{251}+y_1^{246}+y_1^{244}+y_1^{243}+y_1^{242}+y_1^{241}+y_1^{238}+$ $y_1^{237}+y_1^{235}+y_1^{234}+y_1^{233}+y_1^{231}+y_1^{230}+y_1^{225}+y_1^{222}+y_1^{221}+y_1^{220}+y_1^{212}+y_1^{210}+y_1^{208}+y_1^{207}+y_1^{206}+y_1^{205}+y_1^{199}+y_1^{198}+y_1^{197}+$ $y_1^{193}+y_1^{191}+y_1^{190}+y_1^{189}+y_1^{188}+y_1^{187}+y_1^{185}+y_1^{184}+y_1^{180}+y_1^{179}+y_1^{177}+y_1^{176}+y_1^{175}+y_1^{174}+y_1^{170}+y_1^{169}+y_1^{167}+y_1^{166}+y_1^{165}+$ $y_1^{162}+y_1^{160}+y_1^{159}+y_1^{158}+y_1^{156}+y_1^{155}+y_1^{154}+y_1^{148}+y_1^{146}+y_1^{142}+y_1^{141}+y_1^{139}+y_1^{138}+y_1^{137}+y_1^{135}+y_1^{131}+y_1^{130}+y_1^{129}+y_1^{128}+$ $y_1^{126}+y_1^{125}+y_1^{123}+y_1^{122}+y_1^{120}+y_1^{117}+y_1^{115}+y_1^{112}+y_1^{111}+y_1^{110}+y_1^{108}+y_1^{107}+y_1^{105}+y_1^{103}+y_1^{102}+y_1^{101}+y_1^{99}+y_1^{97}+y_1^{96}+$ $y_1^{92}+y_1^{91}+y_1^{86}+y_1^{85}+y_1^{83}+y_1^{82}+y_1^{81}+y_1^{80}+y_1^{79}+y_1^{78}+y_1^{77}+y_1^{76}+y_1^{75}+y_1^{74}+y_1^{72}+y_1^{70}+y_1^{68}+y_1^{67}+y_1^{65}+y_1^{62}+y_1^{61}+y_1^{59}+$ $y_1^{58}+y_1^{57}+y_1^{56}+y_1^{55}+y_1^{54}+y_1^{53}+y_1^{52}+y_1^{51}+y_1^{50}+y_1^{49}+y_1^{48}+y_1^{45}+y_1^{44}+y_1^{43}+y_1^{41}+y_1^{40}+y_1^{39}+y_1^{38}+y_1^{37}+y_1^{36}+y_1^{31}+y_1^{30}+$ $y_1^{27}+y_1^{26}+y_1^{25}+y_1^{24}+y_1^{22}+y_1^{21}+y_1^{20}+y_1^{18}+y_1^{17}+y_1^{16}+y_1^{15}+y_1^{14}+y_1^{13}+y_1^{12}+y_1^{11}+y_1^9+y_1^8+y_1^7+y_1^6+y_1^5+y_1^4+y_1^3+y_1^2+y_1+$ $x_2 \cdot (y_1^{24}+y_1^{23}+y_1^{21}+y_1^{19}+y_1^{17}+y_1^{15}+y_1^{11}+y_1^{10}+y_1^8+y_1^7+y_1^6+y_1^5+y_1^4+y_1^2+1))$

Some MAGMA codes

Here we report some MAGMA functions used during this work.

B.1 Implementation of the Algorithm 3

Here we give the MAGMA code implementing Algorithm 3 in Chapter 5 for the case where the polynomials p_i are all primitive.

```
1
2  ////////////////////////////////
3  ///Preliminary functions//////////
4  ////////////////////////////////
5
6  ////////////////////////////////////////////////
7  ///to print year, month, day, hours, minutes and///
8  ///      seconds of the current moment          ///
9  ////////////////////////////////////////////////
10
11 procedure dateTime(t)
12   // t is seconds since beginning of 1970.
13   // example usage: dateTime(Realtime);
14   days := Floor(t/86400);
15
16   t := t-days*86400;
17   hours := Floor(t/3600);
18   t := t - hours*3600;
19   mins := Floor(t/60);
20   t := t - mins*60;
21   secs := t;
22
23   year := 1970;
24   is_leap := func<y | (y mod 4 eq 0) and (y mod 100 ne 0) select true else false>;
25   day_this := func<y | is_leap(y) select 366 else 365>;
26   while days gt day_this(year) do
27     days -:= day_this(year);
28     year +:= 1;
29   end while;
30   //          J  F  M  A  M  J  J  A  S  O  N  D
```

```
  calendar := [31,28,31,30,31,30,31,31,30,31,30,31];
  month := 1;
  day_that := func<m | m eq 2 and is_leap(year) select 29 else calendar[m]>;
  while days gt day_that(month) do
     days -:= day_that(month);
     month +:=1;
  end while;
  day := days;

  printf "today is %4o/%2o/%2o, it is %2o:%2o'%5.2o'' UTC ",year,month,day,hours,mins,secs;

end procedure;

////////////////////////////////////////////////
///         given an integer d                ////
///it returns a primitive polynomial of degree d///
////////////////////////////////////////////////

function randPrimPol(degree)

  P<x>:=PolynomialRing(GF(2));
  p:= PrimitivePolynomial(GF(2),degree);
  F<b>:=ext<GF(2)|p>;

  repeat
    a:=Random(F);
  until IsPrimitive(a);

  Q<y>:=PolynomialRing(F);
  q:=&*[y-a^(2^i): i in [0..degree-1]];
  return P!q;

end function;

////////////////////////////////////////////////////////
///given a primitive polynomial, it returns its order///
////////////////////////////////////////////////////////

order := function(polynomial)

  return 2^Degree(polynomial) -1;

end function;


//////////////////////////////////////////
```

```
///function generating the exponents e_i///
//////////////////////////////////////////

randomExponents2k := function(orders,weight,targetDegree)

  N := LCM(orders);

  kk :=Floor(Log(2,N/targetDegree));

  wm2 := weight - 2;

  repeat
  // to be more efficient i should break instead of waiting for final loop check

    H := [i eq 1 select 2^kk*Random([1..(targetDegree-wm2+1)])
                   else 2^kk*Random([(Self(i-1)+1)..(targetDegree-wm2+i)])
         : i in [1..wm2]];

    I := [[H[j] mod orders[i]:j in [1..#H]]: i in [1..#orders]];

  until &and[i ne 0: i in &cat(I)] and &and[#I[i] eq #SequenceToSet(I[i]):i in [1..#I]];

  return H, [Sort(I[i]): i in [1..#I]];

end function;

//////////////////////
///Zech's logarithm///
//////////////////////

  Z  := function(i,N)

    if i mod N eq 0 then return -1; end if;

    return  Log(alpha^i+1);

  end function;



  ZZ := function(D,N)

        if #D eq 1 then
          return Z(D[1],N);
        end if;

```

```
123         j1 := $$(Prune(D),N);
124         if j1 eq -1 then return -1; end if;
125         j := D[#D]+j1;
126         if j mod N eq 0 then return -1; end if;
127         return Z(j,N);
128   end function;
129
130 ////////////////////////////
131 ///other basic functions///
132 ////////////////////////////
133
134 function intseqToPoly(E)
135
136
137   GF2X<x> := PolynomialRing(GF(2));
138   // does not work for deg p >~ 2^31
139   return (&+[GF2X ! x^E[i]: i in [1..#E]] + 1 );
140
141 end function;
142
143 /////
144
145 function isRootMultiple(polyexps, root)
146
147   return (&+[root^e: e in polyexps] + 1) eq 0;
148
149 end function;
150
151 ////////////////////////////////
152 //////main procedure/////////////
153 ////////////////////////////////
154
155 load"saveload_util.mgm";
156
157 function computekk(iseq)
158
159   kk := 0;
160
161   while &and[i mod 2^(kk+1) eq 0: i in iseq] do
162     kk +:= 1;
163   end while;
164
165   return kk;
166
167 end function;
168
```

```
algo2kfile := function(pseq, w , D, myfile)

  alpha_ := [* primitiveRoot(p): p in pseq*];
  N_ := [ order(p): p in pseq];

  npol := #pseq;

  string1 := "INPUTS:\npseq\n%o\nw\n%o\ntarget degree\n%o\n";
  string2 := "Log of target degree is %5.2o\n";
  printf string1,pseq,w,D;
  printf string2,Log(2,D*1.0);
  Puts(myfile,Sprintf(string1,pseq,w,D));
  Puts(myfile,Sprintf(string2,Log(2,D*1.0)));

  t0 := Cputime();
  min_degree := LCM(N_);

  count :=1; //dbg;
  repeat
    noerror := true;
    //print count; //dbg
    H, I := randomExponents(N_,w,D);

    J := [ zMultiple(alpha_[r],I[r],N_[r]): r in [1..npol]];
    if &or[j eq -1: j in J] then noerror := false; end if;

    k := CRT(J,N_);
    found := k ne -1;
    count := count +1;
    if noerror and found then
      kk := computekk([k] cat H);
      log_degree := Log(2,k/2^kk);
      if log_degree lt min_degree then
         min_degree := log_degree;
         printf "New minimum degree, its log is: %5.2o\n",min_degree;
         print count,kk,k,H;
         Puts(myfile,
           Sprintf("%5.2o",log_degree) cat " " cat
           IntegerToString(kk) cat " " cat
           IntegerToString(k) cat " " cat
           integerSequenceToString(H)
           );
         tnow := Cputime(t0);
         Days := Floor(tnow/24/60/60);
         tleft := tnow-Days*24*60*60;
         Hours := Floor(tleft/60/60);
```

```
215         tleft := tleft-Hours*60*60;
216         Mins := Floor(tleft/60);
217         tleft := tleft - Mins*60;
218         dateTime(Realtime());
219         printf "It took %o days, %o hours, %o mins, and %5.2o
    seconds\n",Days,Hours,Mins,tleft;
220         print "Expect approximately the same for next advancement";
221       end if;
222     end if;
223   until noerror and found and ((k/2^kk) le D);
224
225   return [k] cat H;
226 end function;
227
228 function isRootMultiple(polyexps, root)
229
230   return (&+[root^e: e in polyexps] + 1) eq 0;
231
232 end function;
233
234
235 ////////////////////
236 //////Example//////
237 ////////////////////
238
239 gf2x<x> := PolynomialRing(GF(2));
240 p1 := PrimitivePolynomial(GF(2),7);
241 p2 := PrimitivePolynomial(GF(2),11);
242 p3 := PrimitivePolynomial(GF(2),13);
243
244 filename := Open("prova.data","w");
245 algo2kfile([p1,p2,p3],5,2^25,filename);
```

B.2 Binary cyclic codes with $t = 3$

The code to produce the list of all binary cyclic codes with $t = 3$ is given below.

```
1 ////////////////////////////
2 ///Some useful functions////
3 ////////////////////////////
4
5 sottoset := function(L)
6  LL := [];
7  i  := 1;
8  for j :=1 to #L do
```

```
  T := [];
  for i := 1 to #L do
   if (i ne j) then
    T := Append(T,L[i]);
   end if;
  end for;
  LL := Append(LL,T);
 end for;
 return LL;
end function;

///////////////

tuttiset :=function(A)
 B:= sottoset(A);
 EE := B;
 flag := 1;
 if (#B[1] eq 1) then
  flag := 0;
 end if;
 while flag eq 1 do
  MM :=[];
  for i := 1 to #B do
   C := sottoset(B[i]);
   if (#C[1] eq 1) then
    flag := 0;
   end if;
   for j := 1 to #C do
    MM := Append(MM,C[j]);
    EE := Append(EE,C[j]);
   end for;
  end for;
  //#C[1];
  //MM;
  B := MM;
 end while;
// print("qui");
 S := { y : y in EE};
 OO :=[];
 for s in S do
  OO := Append(OO,s);
 end for;
 return OO;
end function;

///////////////
```

```
55
56 prodotto := function(L)
57 a := 1;
58 for i:=1 to #L do
59  a := a* L[i];
60 end for;
61 return a;
62 end function;
63
64 ///////////////
65
66 def:= function(g,n)
67  a := RootOfUnity(n,GF(2));
68  for i:=0 to n-1 do
69   if ( Evaluate(g,a^i) eq 0 ) then
70    l:=i; break i;
71   end if;
72  end for;
73  return(l);
74 end function;
75
76
77
78 // We list the codes up to n=61
79
80
81 P<x> := PolynomialRing(GF(2));
82 for m := 2 to 30 do
83  n:=2*m+1;
84  print (" ");
85  print("-----------------------------------------");
86  n;
87  Z := Factorization(x^n+1);
88  a := RootOfUnity(n,GF(2));
89  L :=[];
90  SC := {};
91  i := 1;
92  for i := 1 to #Z do
93   L := Append(L, Z[i,1]);
94  end for;
95  W := tuttiset(L);
96  for i :=1 to #W do
97   LC := {};
98   C := CyclicCode(n,prodotto(W[i]));
99   d := MinimumDistance(C);
100  if ( d eq 7 or d eq 8) then
```

```
    //print("--");
    //d;
    //print("-");
    //for j:=1 to #W[i] do
     //def(W[i][j],n);
    //end for;
    A := { def(p,n) : p in W[i] };
    SC := SC join {A};
   end if;
  end for;
  SC;
end for;
```

B.3 Some classes of binary cyclic codes presented in Chapter 6

We report the codes to produce the classes of binary cyclic codes with length less than 121 which are covered by Theorem 6.2.4, Theorem 6.2.6 and Corollary 6.2.8, the classes of codes with length $n < 105$ which are covered by Theorem 6.2.5, and the classes of codes with length less than 121 which are covered by Theorem 6.3.5 and Theorem 6.3.7.

```

////////////////////////////
///Some useful functions////
////////////////////////////

CodeDefSet:=function(n,S,F);
P<x>:=PolynomialRing(F);
L:=Factorization(x^n+1);
G<b>:=SplittingField(x^n+1);
a:=RootOfUnity(n,G);
A:=[[j: j in [1..#L] | Evaluate(L[j][1],a^i) eq 0]: i in S];
g:= &*[(L[j][1])[1]: j in A];
return CyclicCode(n,g);
end function;

%C:=CodeDefSet(63,{1,3},GF(2));

////////////

DefSet:=function(C,F);
P<x>:=PolynomialRing(F);
n:=Length(C);
g:=GeneratorPolynomial(C);
L:=Factorization(g);
```

```
G<b>:=SplittingField(x^n+1);
a:=RootOfUnity(n,G);
A:={};
for k in [1..#L] do
A:=A join {i: i in [0..n-1]| i eq Min({j: j in [0..n-1] | Evaluate(L[k][1],a^j) eq 0})};
end for;
return A;
end function;

////////////////////////
///Main computations///
////////////////////////

////////////////////////////////////
///codes covered by Theorem 6.2.4///
////////////////////////////////////

A1:=[{CodeDefSet(2*i+1,{1,2^v+1},GF(2)): v in [1..Floor(Log(2,2*i-1))]| (GCD(2^v-1,2*i+1)
    eq 1) and
MinimumDistance(CodeDefSet(2*i+1,{1,2^v+1},GF(2))) in {5,6}}: i in [3..60]];

A1SetL:=[[[{Length(Cod)},DefSet(Cod,GF(2))]: Cod in A1[i]]: i in [1..#A1]| A1[i] ne {}];


////////////////////////////////////
///codes covered by Theorem 6.2.5///
////////////////////////////////////

A1:=[{CodeDefSet(2*i+1,{1,l,s-(2*l),s-l,s},GF(2)): s in [0..2*i], l in [1..2*i]|
    (GCD(s-2*l,2*i+1) eq 1) and (GCD(l, 2*i+1) eq 1) and
    MinimumDistance(CodeDefSet(2*i+1,{1,l,s-2*l,s-l,s},GF(2))) in {5,6}}: i in [3..60]];

A1SetL:=[[[{Length(Cod)},DefSet(Cod,GF(2))]: Cod in A1[i]]: i in [1..#A1]| A1[i] ne {}];

////////////////////////////////////
///codes covered by Theorem 6.2.6///
////////////////////////////////////

A1:=[{CodeDefSet(2*i+1,{1,ExactQuotient(2*i,2^j)},GF(2)): j in [1..Floor(Log(2,2*i-1))] |
    (2^j in Divisors(2*i)) and (3 notin Divisors(2*i+1)) and (2^j ne 2*i) and
    MinimumDistance(CodeDefSet(2*i+1,{1,ExactQuotient(2*i,2^j)},GF(2))) in {5,6}}: i in
    [3..60]];

A1SetL:=[[[{Length(Cod)},DefSet(Cod,GF(2))]: Cod in A1[i]]: i in [1..#A1]| A1[i] ne {}];


```

```
66 //////////////////////////////////////
67 ///codes covered by Corollary 6.2.8///
68 //////////////////////////////////////
69
70 A1:=[{CodeDefSet(2*i+1,{1,2^j,2^j-1,2^j+2},GF(2)): j in [2..Floor(Log(2,2*i-1))] |
       GCD(2^j-2, 2*i+1) eq 1
71 and MinimumDistance(CodeDefSet(2*i+1,{1,2^j,2^j-1,2^j+2},GF(2))) in {5,6}}: i in [3..60]];
72
73 A1SetL:=[[[{Length(Cod)},DefSet(Cod,GF(2))]: Cod in A1[i]]: i in [1..#A1]| A1[i] ne {}];
74
75
76
77 ////////////////////////////////////
78 ///codes covered by Theorem 6.3.5///
79 ////////////////////////////////////
80
81
82 A1:=[{CodeDefSet(2*k+1,{1,i,i+1,i+2,i+3,i+4},GF(2)): i in [0..2*k-1] | ((i mod 2*k+1) ne
       0) and ((i+2 mod 2*k+1) ne 0) and
       MinimumDistance(CodeDefSet(2*k+1,{1,i,i+1,i+2,i+3,i+4},GF(2))) in {7,8}}: k in
       [3..60]];
83
84
85 A1SetL:=[[[{Length(Cod)},DefSet(Cod,GF(2))]: Cod in A1[i]]: i in [1..#A1]| A1[i] ne {}];
86
87
88 ////////////////////////////////////
89 ///codes covered by Theorem 6.3.7///
90 ////////////////////////////////////
91
92
93 A1:=[{CodeDefSet(2*k+1,{1,3,2^i+2^j,2^j-2^i,2^j-2^(i+1)},GF(2)): i in
       [1..Floor(Log(2,2*i-1))], j in [i+2..Floor(Log(2,2*i-1))]| j ge i+2 and
94 MinimumDistance(CodeDefSet(2*k+1,{1,3,2^i+2^j,2^j-2^i,2^j-2^(i+1)},GF(2))) in {7,8}}: k in
       [3..60]];
95
96 A1SetL:=[[[{Length(Cod)},DefSet(Cod,GF(2))]: Cod in A1[i]| Dimension(Cod) ne 0]: i in
       [1..#A1]| A1[i] ne {}];
```

Bibliography

[ABF07] D. Augot, M. Bardet, and J.-C. Faugère, *On formulas for decoding binary cyclic codes*, Proc. of ISIT 2007, 2007, pp. 2646–2650.

[ABF09] D. Augot, M. Bardet, and J.-C. Faugere, *On the decoding of binary cyclic codes with the newton identities*, Journal of Symbolic Computation **44** (2009), no. 12, 1608–1625.

[AH99] L. M. Adleman and M.-D. A. Huang, *Function field sieve method for discrete logarithms over finite fields*, Information and Computation **151** (1999), no. 1, 5–16.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Annals of Mathematics (2004), 781–793.

[BCP97] W. Bosma, J. Cannon, and C. Playoust, *The Magma algebra system I: The user language*, Journal of Symbolic Computation **24** (1997), no. 3, 235–265.

[BES13] E. Ballico, M. Elia, and M. Sala, *On the evaluation of multivariate polynomials over finite fields*, Journal of Symbolic Computation **50** (2013), 255–262.

[BGJT14] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé, *A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic*, Advances in Cryptology–Eurocrypt 2014, Springer, 2014, pp. 1–16.

[Bla03] R. E. Blahut, *Algebraic codes for data transmission*, Cambridge university press, 2003.

[Blu10] SIG Bluetooth, *Bluetooth specification*, 2010.

[BM06] L. M. Bazzi and S. K. Mitter, *Some randomized code constructions from group actions*, IEEE Trans. on Inf. Th. **52** (2006), 3210–3219.

[BMvT78] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg, *On the inherent intractability of certain coding problems*, IEEE Trans. on Inf. Th. **24** (1978), no. 3, 384–386.

[BN90] N. Bruck and M. Naor, *The hardness of decoding linear codes with preprocessing*, IEEE Trans. on Inf. Th. **36** (1990), 381–385.

[BP14] R. Barbulescu and C. Pierrot, *The multiple number field sieve for medium-and high-characteristic finite fields*, LMS Journal of Computation and Mathematics **17** (2014), no. A, 230–246.

[BRC60] R. C. Bose and D. K. Ray-Chaudhuri, *On a class of error correcting binary group codes*, Information and Control **3** (1960), 68–79.

[Cas89] G. Castagnoli, *On the asymptotic badness of cyclic codes with block-lengths composed from a fixed set of prime factors*, LNCS (Berlin / Heidelberg), vol. 357, Springer, 1989, pp. 164–168.

[Che69] C.-L. Chen, *Some results on algebraically structured error-correcting codes*, 1969.

[Chi64] R. T. Chien, *Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes*, Information Theory, IEEE Transactions on **10** (1964), no. 4, 357–363.

[CJM02] P. Chose, A. Joux, and M. Mitton, *Fast correlation attacks: An algorithmic point of view*, Advances in Cryptology—EUROCRYPT 2002, Springer, 2002, pp. 209–221.

[CJS01] V. V. Chepyzhov, T. Johansson, and B. Smeets, *A simple algorithm for fast correlation attacks on stream ciphers*, Fast Software Encryption, Springer, 2001, pp. 181–195.

[CL10] Y. Chang and C.-D. Lee, *Algebraic decoding of a class of binary cyclic codes via lagrange interpolation formula*, Information Theory, IEEE Transactions on **56** (2010), no. 1, 130–139.

[CLF12] Y. Chang, C.-D. Lee, and K. Feng, *Multivariate interpolation formula over finite fields and its applications in coding theory*, arXiv preprint arXiv:1209.1198 (2012).

[CM02] M. Caboara and T. Mora, *The Chen-Reed-Helleseth-Truong decoding algorithm and the Gianni-Kalkbrenner Gröbner shape theorem*, Appl. Algebra Engrg. Comm. Comput. **13** (2002), no. 3, 209–232.

[CMSvS91] G. Castagnoli, J. L. Massey, P. A. Schoeller, and N. von Seeman, *On repeated-root cyclic codes*, IEEE Trans. on Inf.. Th. **37** (1991), 337–342.

[Coo71] S. A. Cook, *The complexity of theorem-proving procedures*, ACM symposium on theory of computing – STOC '71, ACM Press, 1971, pp. 151–158.

[Coo90] A. B. III Cooper, *Direct solution of BCH decoding equations*, Comm., Cont. and Sign. Proc. (1990), 281–286.

[Coo91] A. B. Cooper, *Finding BCH error locator polynomials in one step*, Electronic Letters **27** (1991), no. 22, 2090–2091.

[Cop84] D. Coppersmith, *Fast evaluation of logarithms in fields of characteristic two*, IEEE Trans. on Inf. Th. **30** (1984), no. 4, 587–594.

[CRHT94a] X. Chen, I. S. Reed, T. Helleseth, and K. Truong, *General Principles for the Algebraic Decoding of Cyclic Codes*, IEEE Trans. on Inf. Th. **40** (1994), 1661–1663.

[CRHT94b] X. Chen, I. S. Reed, T. Helleseth, and T. K. Truong, *Algebraic decoding of cyclic codes: a polynomial ideal point of view*, Finite fields, Contemp. Math., vol. 168, Amer. Math. Soc., 1994, pp. 15–22.

[CRHT94c] X. Chen, I. S. Reed, T. Helleseth, and T. K. Truong, *Use of Gröbner bases to decode binary cyclic codes up to the true minimum distance*, IEEE Trans. on Inf. Th. **40** (1994), no. 5, 1654–1661.

[CT00] A. Canteaut and M. Trabbia, *Improved fast correlation attacks using parity-check equations of weight 4 and 5*, Advances in Cryptology—EUROCRYPT 2000, Springer, 2000, pp. 573–588.

[CT12] T. M. Cover and J. A. Thomas, *Elements of information theory*, John Wiley & Sons, 2012.

[CTR+03] Y. Chang, T.-K. Truong, I. S. Reed, H.Y. Cheng, and C.-D. Lee, *Algebraic decoding of* $(71, 36, 11)$, $(79, 40, 15)$, *and* $(97, 49, 15)$ *quadratic residue codes*, IEEE transactions on communications **51** (2003), no. 9, 1463–1473.

[DLC07] F. Didier and Y. Laigle-Chapuy, *Finding low-weight polynomial multiples using discrete logarithm*, Information Theory, 2007. ISIT 2007. IEEE International Symposium on, IEEE, 2007, pp. 1036–1040.

[Eli87] M. Elia, *Algebraic decoding of the (23, 12, 7) golay code*, Information Theory, IEEE Transactions on **33** (1987), no. 1, 150–151.

[For65] G. D. Forney, *On decoding BCH codes*, IEEE Trans. on Inf. Th. **11** (1965), 549–557.

[FT94] G.-L. Feng and K. K. Tzeng, *A new procedure for decoding cyclic and bch codes up to actual minimum distance*, Information Theory, IEEE Transactions on **40** (1994), no. 5, 1364–1374.

[G+82] S. W. Golomb et al., *Shift register sequences*, Aegean Park Press, 1982.

[Gal63] R. Gallager, *Low-Density Parity-Check Codes*, Ph.D. thesis, Massachusetts Institute of Technology, 1963.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to NP-completeness*, WH Freeman & Co., San Francisco (1979).

[GKZ14] R. Granger, T. Kleinjung, and J. Zumbrägel, *On the powers of 2*, IACR Cryptology ePrint Archive **2014** (2014), 300.

[Gol96] J. D. Golić, *Computation of low-weight parity-check polynomials*, Electronics Letters **32** (1996), no. 21, 1981–1982.

[GPS04] C. Gehrmann, J. Persson, and B. Smeets, *Bluetooth security*, Artech house, 2004.

[GV05] V. Guruswami and A. Vardy, *Maximum-likelihood decoding of Reed-Solomon codes is NP-hard*, Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2005, pp. 470–478.

[Har09] D. Harvey, *Faster polynomial multiplication via multipoint Kronecker substitution*, Journal of Symbolic Computation **44** (2009), no. 10, 1502–1510.

[HRTC01] R. He, I. S. Reed, T.-K. Truong, and X. Chen, *Decoding the (47, 24, 11) quadratic residue code*, Information Theory, IEEE Transactions on **47** (2001), no. 3, 1181–1186.

[Hub90] K. Huber, *Some comments on Zech's logarithms*, IEEE Transactions on Information Theory **36** (1990), no. 4, 946–950.

[HV95] J. Hong and M. Vetterli, *Simple algorithms for BCH decoding*, Communications, IEEE Transactions on **43** (1995), no. 8, 2324–2333.

[JJ99a] T. Johansson and F. Jönsson, *Fast correlation attacks based on turbo code techniques*, Advances in Cryptology—CRYPTO'99, Springer, 1999, pp. 181–197.

[JJ99b] ———, *Improved fast correlation attacks on stream ciphers via convolutional codes*, Advances in Cryptology—EUROCRYPT'99, Springer, 1999, pp. 347–362.

[JL06] A. Joux and R. Lercier, *The function field sieve in the medium prime case*, Advances in Cryptology-EUROCRYPT 2006, Springer, 2006, pp. 254–270.

[Jou14] A. Joux, *A new index calculus algorithm with complexity L(1/4+ o (1)) in small characteristic*, Selected Areas in Cryptography–SAC 2013, Springer, 2014, pp. 355–379.

[Jun05] P. Junod, *Statistical cryptanalysis of block ciphers*, Ph.D. thesis, Citeseer, 2005.

[Kas74] T. Kasami, *A Gilbert-Varshamov bound for quasi-cycle codes of rate* $1/2$ *(corresp.)*, IEEE Transactions on Information Theory (1974), 679–679.

[Knu81] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, vol. 2, Addison-Wesley, 1981.

[KP94] G. J. Kühn and W. T. Penzhorn, *Using Zech's logarithm to find low-weight parity checks for linear recurring sequences*, Communications and Cryptography, Springer, 1994, pp. 221–225.

[Kra22] M. Kraitchik, *Théorie des nombres, vol. 1*, Gauthier-Villars, Paris (1922).

[Kro82] L. Kronecker, *Grundzüge einer arithmetischen theorie der algebraischen grössen*, G. Reimer, 1882.

[LCCC10] C.D. Lee, Y. Chang, H.H. Chang, and J.H. Chen, *Unusual general error locator polynomial for the* $(23, 12, 7)$ *Golay code*, Communications Letters, IEEE **14** (2010), no. 4, 339–341.

[LCJM12] C.-D. Lee, Y. Chang, M.-H. Jing, and J.-H. Miao, *New method of predetermining unified unknown syndrome representations for decoding binary cyclic codes*, IET Communications **6** (2012), no. 18, 3339–3349.

[LCTC10] C.-D. Lee, Y. Chang, T.-K. Truong, and Y.-H. Chen, *More on general error locator polynomials for a class of binary cyclic codes*, Information Theory and its Applications (ISITA), 2010 International Symposium on, IEEE, 2010, pp. 273–277.

[Lee11] C.-D. Lee, *Weak general error locator polynomials for triple-error-correcting binary Golay code*, IEEE communications letters **15** (2011), no. 8, 857–859.

[LJM12] C.-D. Lee, M.-H. Jing, and J.-H. Miao, *Algebraic decoding of a class of ternary cyclic codes*, Signal Processing, Communication and Computing (ICSPCC), 2012 IEEE International Conference on, IEEE, 2012, pp. 331–336.

[LN97] R. Lidl and H. Niederreiter, *Finite Fields*, Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1997.

[LV04] Y. Lu and S. Vaudenay, *Faster correlation attack on Bluetooth keystream generator E0*, Advances in Cryptology–CRYPTO 2004, Springer, 2004, pp. 407–425.

[LW67] S. Lin and E. J. Weldon, *Long BCH codes are bad*, Information Control **11** (1967), 445–451.

[LY97] P. Loustaunau and E. V. York, *On the decoding of cyclic codes using Gröbner bases*, AAECC **8** (1997), no. 6, 469–483.

[Mas69] J. L. Massey, *Shift-register synthesis and BCH decoding*, IEEE Trans. on Inf. Th. **15** (1969), 122–127.

[MBG+13] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of finite fields*, vol. 199, Springer Science & Business Media, 2013.

[MFI01] M. J. Mihaljević, M. P. C. Fossorier, and H. Imai, *A low-complexity and high-performance algorithm for the fast correlation attack*, Fast Software Encryption, Springer, 2001, pp. 196–212.

[MM07] G. L. Mullen and C. Mummert, *Finite fields and applications*, vol. 41, American Mathematical Soc., 2007.

[MO09] T. Mora and E. Orsini, *Decoding cyclic codes: the Cooper philosophy*, Gröbner Bases, Coding, and Cryptography, Springer, 2009, pp. 69–91.

[Mor03] T. Mora, *Solving polynomial equation systems i: The Kronecker-Duval philosophy, volume 88 of*, Encyclopedia of Mathematics and its Applications (2003).

[MOS12] C. Marcolla, E. Orsini, and M. Sala, *Improved decoding of affine-variety codes*, Journal of Pure and Applied Algebra **216** (2012), no. 7, 1533–1565.

[MP13] G. L. Mullen and D. Panario, *Handbook of Finite Fields*, CRC Press, 2013.

[MPW06] C. Martinez-Perez and W. Willems, *Is the class of cyclic codes asymptotically good?*, IEEE Trans. on Inf. Th. **52** (2006), no. 2, 696–700.

[MS77] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes. I and II*, North-Holland Publishing Co., Amsterdam, 1977.

[MS88] W. Meier and O. Staffelbach, *Fast correlation attacks on stream ciphers*, Advances in cryptology - EUROCRYPT-88, Springer, 1988, pp. 301–314.

[MS89] W. Meier and O. Staffelbach, *Fast correlation attacks on certain stream ciphers*, Journal of Cryptology **1** (1989), no. 3, 159–176.

[MS12] L. Minder and A. Sinclair, *The extended k-tree algorithm*, Journal of cryptology **25** (2012), no. 2, 349–382.

[MW86] G. Mullen and D. White, *A polynomial representation for logarithms in GF (q)*, Acta arithmetica **3** (1986), no. 47, 255–261.

[OS05] E. Orsini and M. Sala, *Correcting errors and erasures via the syndrome variety*, J. Pure Appl. Algebra **200** (2005), 191–226.

[OS07] ———, *General error locator polynomials for binary cyclic codes with* $t \leq 2$ *and* $n < 63$, IEEE Trans. on Inf. Th. **53** (2007), 1095–1107.

[PBH98] V. Pless, R. A. Brualdi, and W. C. Huffman, *Handbook of coding theory*, Elsevier Science Inc., 1998.

[PH78] S. C. Pohlig and M. E. Hellman, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, Information Theory, IEEE Transactions on **24** (1978), no. 1, 106–110.

[PHB98] V. S. Pless, W. C. Huffman, and R. A. Brualdi (eds.), *Handbook of Coding Theory. Vol. I, II*, North-Holland, Amsterdam, 1998.

[Pol78] J. M. Pollard, *Monte Carlo methods for index computation mod p*, Mathematics of computation **32** (1978), no. 143, 918–924.

[Pra57] E. Prange, *Cyclic Error-Correcting Codes in Two Symbols*, Technical Report AFCRRC TN-57-103, Air Force Cambridge Research Center, Cambridge, MA, 1957.

[PST16] P. Peterlongo, M. Sala, and C. Tinnirello, *A discrete logarithm-based approach to compute low-weight multiples of binary polynomials*, Finite Fields and Their Applications **38** (2016), 57–71.

[PT78] G. Promhouse and S. E. Tavares, *The minimum distance of all binary cyclic codes of odd lengths from* 69 *to* 99, Information Theory, IEEE Transactions on **24** (1978), no. 4, 438–442.

[PTS14] P. Peterlongo, C. Tinnirello, and M. Sala, *Low-weight common multiples of binary primitive polynomials through discrete logarithms*, Book of Abstracts, YACC 2014, 2014, pp. 103–107.

[PW72] W. W. Peterson and Jr. E. J. Weldon, *Error-correcting codes*, second ed., The M.I.T. Press, Cambridge, Mass.-London, 1972.

[RS60] I. S. Reed and G. Solomon, *Polynomial codes over certain finite fields*, J. Soc. Indust. Appl. Math. **8** (1960), 300–304.

[Rue86] R. A. Rueppel, *Stream ciphers*, Analysis and Design of Stream Ciphers, Springer, 1986, pp. 5–16.

[Sch00] O. Schirokauer, *Using number fields to compute logarithms in finite fields*, Mathematics of Computation of the American Mathematical Society **69** (2000), no. 231, 1267–1283.

[SER11] D. Schipani, M. Elia, and J. Rosenthal, *On the decoding complexity of cyclic codes up to the BCH bound*, Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on, IEEE, 2011, pp. 835–839.

[Sha48] C. E. Shannon, *A mathematical theory of communication*, Bell System Tech. J. **27** (1948), 379–423, 623–656.

[Sie84] T. Siegenthaler, *Correlation-immunity of nonlinear combining functions for cryptographic applications (corresp.)*, Information Theory, IEEE Transactions on **30** (1984), no. 5, 776–780.

[Sie85] T. Siegenthaler, *Decrypting a class of stream ciphers using ciphertext only*, Computers, IEEE Transactions on **100** (1985), no. 1, 81–85.

[SS81] R. Schroeppel and A. Shamir, *A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for certain NP-Complete problems*, SIAM journal on Computing **10** (1981), no. 3, 456–464.

[TCCL05] T.-K. Truong, Y. Chang, Y.-H. Chen, and C.-D. Lee, *Algebraic decoding of $(103, 52, 19)$ and $(113, 57, 15)$ quadratic residue codes*, IEEE transactions on communications **53** (2005), no. 5, 749–754.

[TSS+08] T.-K. Truong, P.-Y. Shih, W.-K. Su, C.-D. Lee, and Y. Chang, *Algebraic decoding of the $(89, 45, 17)$ quadratic residue code*, Information Theory, IEEE Transactions on **54** (2008), no. 11, 5005–5011.

[Var97] A. Vardy, *Algorithmic complexity in coding theory and the minimum distance problem*, Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997, pp. 92–109.

[vL91] J. H. van Lint, *Repeated-root cyclic codes*, Information Theory, IEEE Transactions on **37** (1991), no. 2, 343–345.

[Wag02] D. Wagner, *A generalized birthday problem*, Advances in cryptology—CRYPTO 2002, Springer, 2002, pp. 288–304.