Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Performance of cyclic redundancy checks using error-correcting codes (Dutch title: Performantie van Cyclische Redundantie Checks in combinatie met fout-verbeterende codes

Thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

Jochem de Jong
5169046

Delft, The Netherlands
June 29, 2023

TU Delft

BSc thesis Applied Mathematics

Performance of cyclic redundancy checks using error-correcting codes (Dutch title: Performantie van Cyclische Redundantie Checks in combinatie met fout-verbeterende codes

Jochem de Jong
5169046

Delft University of Technology

Supervisor

Dr. Ir. J.H. Weber

Thesis committee
Dr. J.A.M. de Groot

June 29, 2023, Delft

TU Delft

Preface

This thesis that I have written marks the final part of the Bachelor Applied Mathematics at Delft University of Technology. In this thesis an application of coding theory is discussed which is quite important in many sectors of mathematics, computer science and electrical engineering. Prior to this project I followed the courses Linear Algebra 1, Algebra 1 and the minor course Computer Organisation, from which grew the interest in applications of vector spaces, group theory, modular arithmetic and storing data. This, thereafter led to choosing the elective course Applied Algebra: Codes. This course discussed the fallibility of message transmission and the power of error-correcting codes using Linear Algebra. After finishing these courses I knew that I wanted to do a project related to these subjects, in particular related to coding theory, because coding theory is quite vast and there is a lot more to be studied, given its many applications in telecommunication, for instance in 5G [1] .

In the course there is an extensive focus on the performances of codes with respects to error correction, this power, once again became clear in this project where the focus will be on cyclic redundancy checks in combination with such an error-correcting code and how this impacts the probability that an undetected error occurs. Cyclic redundancy checks also play an important role in telecommunication systems.

I would like to use this part of the thesis to express my gratitude towards my supervisor Dr. Jos Weber for guiding me in the right direction and giving me insights into the project and useful background information.

Jochem de Jong
Delft, June 29, 2023

Abstract

Every day an extremely large amount of messages get sent over the internet, radio, WiFi, etc. These messages need to be encoded in order to protect the important data. However just like many of these systems messages can get corrupted over a certain channel due to noise, fluctuations in power or temperature etc. [2] Therefore cyclic redundancy checks (CRC), which are bits that are appended to message words to protect the data, are used to detect these errors. This is done by using CRC polynomials. An occurring undetected error is possible in any case, therefore we need to study closely the probability of an undetected error occurring, that is the probability that the received, erroneous data is accepted as the transmitted message. The code words that are the results of adding cyclic redunancy checks (CRC) can also be combined with an error-correcting codes like BCH codes. As shown in [3] combining CRC polynomials with error-correcting can reduce the previously mentioned probability, but one CRC polynomial can perform better. In this thesis these CRC polynomials are studied in combination with certain different codes and are evaluated based on their performance with respects to their undetected error probability. Two examples of codes are given, in the first example a CRC polynomial is found that does not perform well in the system without using an error-correcting code, but performs the best in the system with an error-correcting code. An analysis is done which shows that by looking at consecutive roots of the generator polynomial, optimal CRC polynomials can be found. The second example uses a different code and finds another optimal CRC polynomial.

Contents

Chapter 1

Introduction

1.1 Motivation

The worst outcome in telecommunication is when mistakes are made but not detected and thus go through as though it were correct. However due to fallibility this cannot be prevented. It is vital that communication is done without interference or corruption. We can however detect failure by using cyclic redundancy checks (CRC). Systems that use CRC are implemented in many sectors of telecommunication. The disadvantage of CRC systems however is that errors can only be detected but not corrected since in case of failure a resubmission request is made. However error-correcting codes can be used together with CRC systems to be able to correct the errors that are made. In this case the probability that errors are undetected decreases because this scheme uses two steps for decoding, and thus there is a higher probability of detecting the error. This makes it a useful implementation but it gives rise to re-evaluate the respective performance of CRC polynomials when comparing both implementations, one without the error-correcting code and the other with an error-correcting code. In this thesis these two systems are compared in terms of CRC polynomial evaluation.

1.2 Thesis Statement

The paper of K.A. Abdel-Ghaffar named "CRC in Coded Schemes with Bounded-Distance Decoding" [3] is what gives rise to the purpose of this thesis. In this paper it is discussed that some CRC polynomials perform well in the regular CRC systems, whereas in combination with an error-correcting code they perform less optimal. Here a selection method is also given in finding an optimal CRC polynomial by selecting those polynomials that generate the fewest amount of lower weight code words. This however is a selection based on continuously dividing polynomials which can take a long time. In this thesis the focus will generally be on the polynomials themselves and specifically the amount of consecutive roots that they have to be able to say something about the minimal weight before looking at the actual weight distribution and aims to find a method that finds optimal CRC polynomials based on comparing the amount of consecutive roots.

1.3 Organization

In this section a general summary of the chapters is given of this thesis

Chapter 2: Prerequisites
In this chapter the back ground information on error-correcting codes is given and the set up is given

for the two CRC systems that will be evaluated later and the formulas that are used to calculated the undetected error probability are given, which are later used to find the results.

Chapter 3: CRC systems with BCH codes
This chapter gives the main results that are found in this thesis. Two different BCH codes are used to compare CRC polynomial performance and a method is given to explain why some CRC polynomials perform better.

Chapter 4: Conclusions and Discussion
This chapter gives all the conclusions based on the results of this thesis and a discussion is done with respect to future research.

Chapter 2

Prerequisites

In this chapter some basic concepts are explained before looking into the different kind of CRC polynomials. In Section 2.1 we will give the assumptions on the channel that we will use throughout this thesis to avoid any ambiguity. In Section 2.2 the relevant concepts of coding are given, followed by the explanation in full detail of CRC encoding and decoding and the two different systems in Section 2.3 and 2.4 and we will give the corresponding formulas for the probability of an undetected error occurring.

2.1 Channel Assumptions

Before we can look into CRC polynomials the assumptions on the channel need to be clear to not create any ambiguity or unclarity. First we assume all transferred messages to be binary words of fixed length, so bits are only equal to 0's or 1's. We assume that no insertions or deletions are created during transmission, that is no bits are deleted or inserted resulting in a different length of code words. Data corruption can only occur through bit flipping which is the process of 0's changing in 1's and vice versa. Throughout the thesis we use a binary symmetrical channel, which means that these respective probabilities are equal. The probability of a bit flipping will be denoted by p, this in turn means that $1 - p$ is the general probability of a bit not changing. This process of bit flipping is what makes the process vulnerable to undetected errors, which is what we want to prevent and therefore we need to reduce the probability of an undetected error occurring. This can also be seen in Figure 2.1.

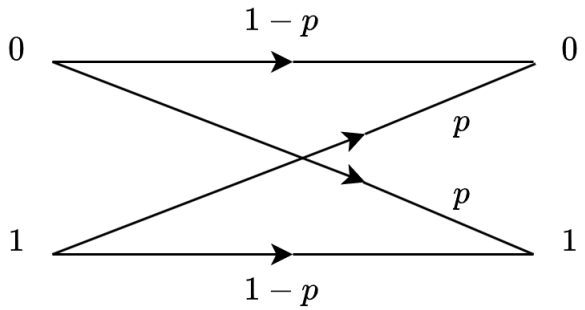

Figure 2.1: A symmetrical channel with probability p that a bit flips

We will also assume that these bit flips are independent on each other, i.e., the event that a bit flips (and thus an error occurs) does not depend on the event that a neighbouring bit (or any other

bit) flips. It is a realistic process that these probabilities are not independent, because errors can come in bursts, for example and can definitely cause significant data loss, as explained in [6]. The problem of burst errors can be resolved with a process called interleaving, however that will be outside the scope of this thesis.

2.2 Coding Theory

In this section we will first give an introduction on linear *codes*. Linear codes are used because they have special linear algebra properties. Codes are sets of binary words that have a fixed length, in particular cyclic codes, which are a special type of linear codes are used widely because there a lot of properties that come from linear algebra that can be used to determine the error-correcting performance of the code. There are many different kind of cyclic codes such as BCH codes, that will also be used in this thesis.

2.2.1 Definition Linear Code

Let $(\mathbb{F}_2)^n$ be the set of binary words of length n, i.e:

$$(\mathbb{F}_2)^n = \{\mathbf{a} = a_0 a_1 \cdots a_{n-1} \text{ such that } a_0, \ldots, a_{n-1} \in \{0, 1\}\}$$

A code $\mathcal{C}$ is a subset of $(\mathbb{F}_2)^n$. Note that the words from $(\mathbb{F}_2)^n$ can equivalently be regarded as (row) vectors of length n. This immediately gives rise to use linear algebra, therefore we first define linear codes.

Let $\mathcal{C} \subset (\mathbb{F}_2)^n$ be a code. $\mathcal{C}$ is linear if all combinations of code words are also code words, i.e., a linear code is contained under addition and multiplication with a scalar, with the scalar being either a 0 or a 1 since we use a binary field. This immediately implies that all linear codes contain the zero-vector, i.e., the word that has all bits equal to 0. Notice that the code words can equivalently be represented as (row) vectors, thus we can regard a linear code as a linear vector space. Hence we can find a basis for the code, that is, a set of linearly independent words (vectors) that span the whole code. The amount of code words in a basis for $\mathcal{C}$ is given by the parameter k, which is the dimension of the code. All codes have another parameter which is the minimal distance d. d is such that $d(\mathbf{x}, \mathbf{y}) \geq d$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, where $d(\mathbf{x}, \mathbf{y})$ is equal to the amount of bits in which $\mathbf{x}$ and $\mathbf{y}$ differ. The handy property of linear codes is that d is equal to the code word with minimal weight of $\mathcal{C}$, i.e., the code word that has the least amount of bits equal to 1. We write that $\mathcal{C}$ is a $[n, k, d]$ code with length n, dimension k and minimal distance d.

2.2.2 Polynomial Representation

In Section 2.2.1 we have given the defintion of a linear code. As stated, each binary word of length n can be represented as a (row) vector of length n. An equivalent way of writing a code word is by writing its polynomial representation. For a binary word $\mathbf{c} = c_0 c_1 \ldots c_{n-1}$ of length n, its polynomial representation is given by

$$c(x) = \sum_{i=0}^{n-1} c_i x^i = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}$$

We denote its degree by $\deg(c(x))$ and note that $\deg(c(x)) \leq n-1$, since the coefficients $c_0, \ldots, c_{n-1}$ can all be equal to either 0 or 1.

2.2.3 Polynomial code

Now that we have given the polynomial representation we can define a polynomial code. A polynomial code is a special type of linear codes. Let $g(x)$ be a polynomial. Suppose that $\mathcal{C}$ is a linear code. Then $\mathcal{C}$ is called a polynomial code if for each code word $\mathbf{c} \in \mathcal{C}$, $c(x)$ is divisible by $g(x)$. Suppose that $g(x)$ has degree l. This means that for $c \in \mathcal{C}$ it holds that $c(x) = a(x)g(x)$ for some binary word $a(x)$ for which $\deg(a(x)) < n - l$. In other words, we can write any cyclic code $\mathcal{C}$ that has $g(x)$ as corresponding polynomial in the following way:

$$\mathcal{C} = \{a(x)g(x) \mid \deg(a(x)) < n - l\}$$

We call $g(x)$ the generator polynomial of $\mathcal{C}$, since it generates the codes.

2.2.4 Cyclic Codes

A cyclic code $\mathcal{C}$ is a linear code that is contained under cyclic shifts. That is, for all code words $\mathbf{c} = c_0c_1 \cdots c_{n-1} \in \mathcal{C}$, its cyclic shift $\pi(\mathbf{c}) = c_{n-1}c_0c_1 \cdots c_{n-2} \in \mathcal{C}$. The handy property is that a cyclic code is generated by a generator polynomial $g(x)$, such that $g(x)$ divides $x^n - 1$, or in other words, cyclic codes are a special type of polynomial codes. This generator polynomial $g(x)$ is used to encode our messages. Suppose a transmitted message $\mathbf{u}$ of length k is sent. This message gets encoded, using code $\mathcal{C}$ to a code word $\mathbf{c}$ of length n, by multiplying its polynomial representation with the generator polynomial $g(x)$. Therefore there is a redundancy of $l = n - k$ bits. $\mathbf{c}$ is the code word that gets sent over the channel, where errors can occur. The resulting word $\tilde{\mathbf{c}}$, then gets decoded over a decoding channel.

2.2.5 Decoding

The decoder that is used is a t-bounded-distance decoder, that works as followed and is described in [3]. Given two words $\mathbf{x}$ and $\mathbf{y}$, the Hamming Distance $d(\mathbf{x}, \mathbf{y})$ between $\mathbf{x}$ and $\mathbf{y}$ is defined as the amount of bits in which $\mathbf{x}$ and $\mathbf{y}$ differ. Let d be the minimal distance between any two code words in $\mathcal{C}$. Suppose now that $\mathbf{c} \in \mathcal{C}$ is the transmitted code word over the channel. The received word $\tilde{\mathbf{c}}$ might be unequal to $\mathbf{c}$ due to possible errors occurring. The word $\tilde{\mathbf{c}}$ is then sent to a decoding channel that is capable of correcting at most $t = \lfloor \frac{d-1}{2} \rfloor$ errors. It finds a code word $\mathbf{a} \in \mathcal{C}$ closest to $\tilde{\mathbf{c}}$ such that the Hamming Distance between $\mathbf{a}$ and $\tilde{\mathbf{c}}$ is at most t. Therefore if at most t errors occur, the decoding is successful. If however more than t errors occur, two things can happen:

- There is another code word $\mathbf{a} \in \mathcal{C}$ within hamming distance less than t to $\tilde{\mathbf{c}}$, and thus the decoded message is other than $\mathbf{c}$, since $\mathbf{c}$ and $\tilde{\mathbf{c}}$ have hamming distance higher than t. Therefore the decoding returns the wrong code word. This results in an *undetected error*.

- The received word $\tilde{\mathbf{c}}$ is not a code word from $\mathcal{C}$ and there are no other code words within distance t to $\tilde{\mathbf{c}}$, therefore the channel is not able to decode the message, in which case resubmission is requested, this results in a *decoding failure*.

In the first of these two cases there is an undetected error, since the channel finds a code word other than $\mathbf{c}$ within t distance of the $\tilde{\mathbf{c}}$, despite the fact that more than t errors occurred. Therefore the message gets wrongfully decoded and thus the message is distorted. In the case of the decoding failure re-submission is requested resulting in no problem since the error is still being detected. Therefore it is highly important to be able to reduce the undetected error probability.

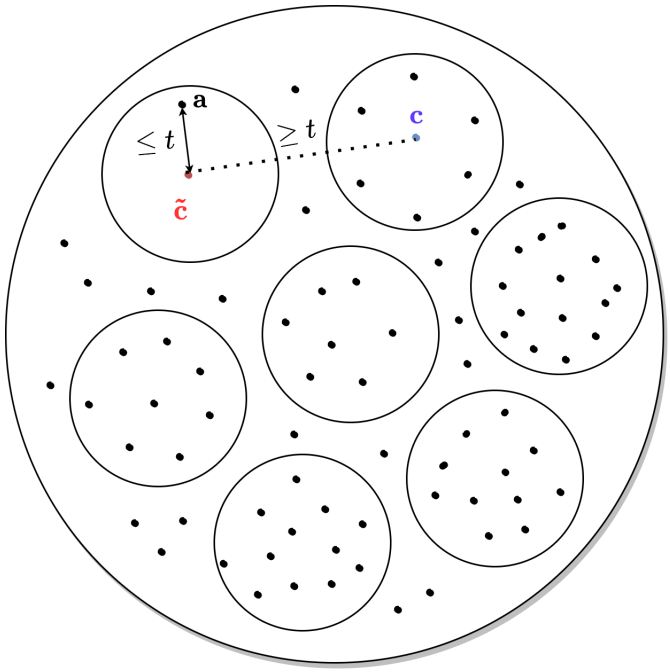

Figure 2.2: Undetected error occurring in a t-bounded-distance decoder

In Figure 2.2 the decoding system is visually described in the case that an undetected error occurs. The larger circle represents $(\mathbb{F}_2)^n$, which is the full collection of binary code words of length n. The middle of the smaller circles represent the code words of $\mathcal{C}$. If the word $\tilde{\mathbf{c}}$ is inside the smaller circle corresponding to $\mathbf{c}$, then at most t errors have occurred and the code word is successfully decoded. If however $\tilde{\mathbf{c}}$ is in a different smaller circle, as visualised in Figure 2.2, it means that more than t errors have occurred and thus the decode result is a code word $\mathbf{a}$ and there is an undetected error. Lastly if $\tilde{\mathbf{c}}$ is outside of any smaller circle it means that it is in $(\mathbb{F}_2)^n \backslash \mathcal{C}$ and thus also more than t errors have occurred, but in this case there is no code word with distance t closest to $\tilde{\mathbf{c}}$ and thus a decode failure occurs and resubmission is requested.

2.3 CRC without error correcting codes

With the use of CRC polynomials in combination with an error-correcting code we wish to reduce the probability of an undetected error occurring. But we have to be careful with our polynomial choice since the performance of CRC polynomials can change respectively when combining the CRC polynomials with an error-correcting code, as is already discussed in [3]. Before we introduce the combination of error-correcting codes we will explain how CRC works when not combining with an error-correcting code. We can view a CRC polynomial $g_{\text{CRC}}(x)$ as a generator polynomial that generates a linear code $\mathcal{C}_{\text{CRC}}$, thus $\mathcal{C}_{\text{CRC}}$ is a polynomial code.

2.3.1 CRC encoding

Suppose $\mathbf{a} = (a_{m-1}, a_{m-2}, \ldots, a_0)$ of is the message of m bits we want to send and that the degree of $g_{\text{CRC}}(x)$ is l. Then $\mathbf{a}$ is encoded to a code word $\mathbf{c} \in \mathcal{C}_{\text{CRC}}$ of $m+l$ bits. This is done by appending l bits, depending on a division. This is done by noting the following:

$$x^l a(x) \equiv r(x) \mod g_{\text{CRC}}(x)$$

for some remainder polynomial $r(x)$, for which $\deg(r) < l$. This remainder is found by dividing $x^l a(x)$ by $g_{\text{CRC}}(x)$, as the modular arithmetic implies. CRC works by calculating this remainder polynomial $r(x)$ and then appending the corresponding bits to the message $a(x)$. In Figure 2.3 the encoding system is visualised.

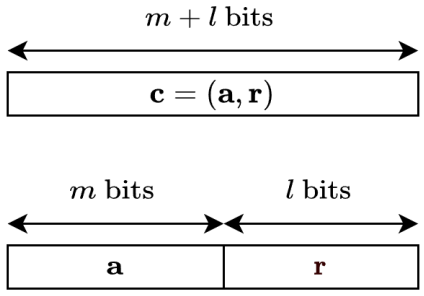

Figure 2.3: Encoding scheme for CRC codes

Before we introduce the decoding scheme we will first show that, this way of encoding the messages is well-defined, that is: $\mathcal{C}_{\text{CRC}}$ forms a polynomial code with generator polynomial $g_{\text{CRC}}(x)$.

Theorem 2.1. *Let* $\mathbf{c}$ *be a CRC-word and let* $g_{CRC}(x)$ *be the corresponding CRC-polynomial with degree* l*, then* $g_{CRC}(x)$ *generates a polynomial code* C_{CRC}*.*

Proof. We only need to show that $g_{\text{CRC}}(x)$ generates the code, by showing that each CRC-word is indeed divisible by $g_{\text{CRC}}(x)$. Let $a = (a_{m-1}, a_{m-2}, \ldots, a_0)$. Therefore, equivalently $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1}$. Note that $x^l a(x) \equiv r(x) \bmod g_{\text{CRC}}(x)$, where $r(x) = (r_{l-1}, r_{l-2}, \ldots, r_0)$. Also note that $\mathbf{c} = (\mathbf{a}, \mathbf{r})$, thus we can write, equivalently in polynomial representation:

$$\begin{aligned} c(x) &= a_{m-1} x^{m+l-1} + a_{m-2} x^{m+l-2} + \cdots + a_0 x^l + r_{l-1} x^{l-1} + r_{l-2} x^{l-2} + \cdots + r_0 \\ &= x^l a(x) + r(x) \end{aligned}$$

But $x^l a(x) + r(x)$ is divisible by $g_{\text{CRC}}(x)$ by definition of $r(x)$, thus c is divisible by $g_{\text{CRC}}(x)$ □

Therefore, $g_{\text{CRC}}(x)$ is indeed the generator polynomal of a cyclic code $\mathcal{C}_{\text{CRC}}$.

2.3.2 CRC decoding

CRC decoding is done solely by checking whether the received word is a CRC-word or not. If the word is a CRC-word, the word is accepted and is the decoded word. The CRC word $\mathbf{c}$ gets sent over the channel and the result is a word $\tilde{\mathbf{c}}$. If $\tilde{\mathbf{c}}$ is divisible by $g_{\text{CRC}}(x)$, the received word is a CRC-word and is thus accepted as the decoded word. If however $\tilde{\mathbf{c}}(x)$ is not divisible by $g_{\text{CRC}}(x)$, an error has occurred and thus a decoding failure occurs and resubmission is requested. If this is not the case the message gets rejected and thus there is a decoding failure, as is visualized in Figure 2.4.

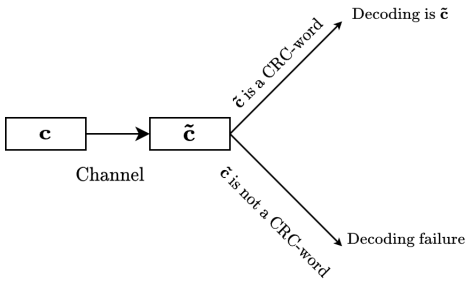

Figure 2.4: Decoding system for CRC codes

2.3.3 Undetected error probability

Remember that the probability that an individual bit flips equals p, as defined in Section 2.1. We will now calculate the probability that an undetected error occurs for the CRC-system. Remember that an undetected occurs in the following case: First a message of m bits is encoded using CRC-encoding and the resulting code word $\mathbf{c}$ of length $m+l$ bits is transmitted through a certain channel. A word $\tilde{\mathbf{c}} \neq \mathbf{c}$ is then received and thus an error has occurred. In the case of an undetected error, $\tilde{\mathbf{c}}$ is a different CRC code word and is wrongly accepted as the transmitted code word. Now, let $A_w^{g_{\text{CRC}}}$ denote the amount of code words of weight w in $\mathcal{C}_{\text{CRC}}$, that is the amount of the words with weight w that are divisible by $g_{\text{CRC}}(x)$. $A_w^{g_{\text{CRC}}}$ are called the *weight enumerators* of $\mathcal{C}_{\text{CRC}}$. We will now give the following theorem:

Theorem 2.2. *Suppose the length of the message words is m. Let $\mathcal{C}_{CRC}$ be the linear code generated by $g_{CRC}(x)$, with $deg(g_{CRC}(x)) = l$ and let $A_w^{g_{CRC}}$ be the weight enumerators of $\mathcal{C}_{CRC}$. Then, for a transmitted code word, and channel error rate p, the probability that an undetected error, P_{ue} occurs is equal to:*

$$P_{ue}(p) = \sum_{w=1}^{m+l} A_w^{g_{CRC}} p^w (1-p)^{m+l-w}$$

Proof. We can assume W.L.O.G. that the transmitted code word is $\mathbf{0}$, which is the zero code word, because $\mathcal{C}_{CRC}$ is a linear code. Let $\tilde{\mathbf{c}}$ be a CRC-word of weight w, that is: $\tilde{\mathbf{c}}$ consists of w bits equal to 1 and $m+l-w$ bits equal to 0. The probability that $\mathbf{0}$ turns into $\tilde{\mathbf{c}}$, is therefore equal to the probability that w bits equal to 0 turn into bits equal to 1 and that the remaining $m+l-w$ bits, do not flip. Thus

$$P(\mathbf{0} \text{ turns into } \tilde{\mathbf{c}}) = p^w (1-p)^{m+l-w}$$

Note that $\tilde{\mathbf{c}}$ was an arbitrary code word of weight w and thus:

$$P(\mathbf{0} \text{ turns into a code word of weight } w) = A_w^{g_{CRC}} p^w (1-p)^{m+l-w} \tag{2.1}$$

Now we only need to sum 2.1 for all weights w and this gives the result. □

2.4 CRC system with error-correcting codes

So far we have only introduced the encoding and decoding system for CRC systems without using an error-correcting code. In this section we will explain how the CRC system works when combining

with an error-correcting code. The system works as follows.

Given a CRC polynomial $g_{\text{CRC}}(x)$, we can encode a message $\mathbf{a}$ to a CRC-word $\mathbf{c}$. Denote the length of $\mathbf{c}$ by k. We can then use a $[n, k, d]$ cyclic code, denote $\mathcal{C}$, to encode $\mathbf{c}$ of length k to a new code word $\mathbf{u}$ of length n. After this, $\mathbf{u}$ is transmitted over the channel and $\tilde{\mathbf{u}}$ is received. A t-bounded distance decoder is then used to decode the messages. If the t-bounded-distance decoder returns a different code word $\mathbf{u}_1$, an undetected error has occurred, but the advantage of this system is that now the message needs to be translated back to a message of k and therefore the resulting word might not be divisible by g_{CRC}, therefore still detecting the error and resubmission of the message is asked.

2.4.1 Undetected error probability

Since we have now introduced an error-correcting code the probabilty that an undetected error occurs is a little more tedious to calculate, since there is a t-bounded distance decoder involved. Let $g_{\mathcal{C}}(x)$ be the generator polynomial generating the polynomial code $\mathcal{C}$. Now we note that $\mathcal{C}$ consists of code words that can be found by multiplying $g_{\mathcal{C}}(x)$ with a message word of length k, which are not all necessarily CRC-words. As defined in [3] we denote $A_w^{g_{CRC}}(\mathcal{C})$ as the amount of code words in $\mathcal{C}$ of weight w that are the result of encoding a CRC-word using $g_{\mathcal{C}}(x)$. These are the code words that cause an undetected error, in other words: if the CRC-word of length k is encoded to $\mathbf{c} \in \mathcal{C}$ and any other code word in $\mathcal{C}$ is received, an undetected error occurs. Thus we need to calculate the probability that this happens. We give the following theorem of a result found by [3]:

Theorem 2.3. *Let $g_{CRC}(x)$ be a CRC-polynomial that encodes messages to CRC words of length k and let $\mathcal{C}$ be the $[n, k, d]$ error-correcting code ($d > 0$) used to encode the CRC-words. Suppose a t-bounded distance decoder is used to decode the received message, then:*

$$P_{ue}(\mathcal{C}, t, p) = \sum_{w=1}^{n} A_w^{g_{CRC}}(\mathcal{C}) \sum_{\substack{t_0, t_1 \geq 0 \\ t_0 + t_1 \leq t}} \binom{w}{t_0} \binom{n-w}{t_1} p^{w+t_1-t_0}(1-p)^{n-w-t_1+t_0} \tag{2.2}$$

Proof. W.L.O.G. we assume that $\mathbf{0}$ is transmitted, which is possible since $\mathcal{C}$ is a linear code. Suppose that the channel causes errors and that $\mathbf{c} \in \mathcal{C}$, with weight $w \geq 1$, is the decoded word which comes from encoding a CRC-word. This means that an undetected error has occurred. Hence the received word through the channel is a word $\tilde{\mathbf{c}}$ which is closer to $\mathbf{c}$ than to $\mathbf{0}$. Notice that t is always strictly smaller than d and thus that $w^* > t$, where w^* is the minimal weight of $\mathcal{C}$, therefore $d(\mathbf{0}, \tilde{\mathbf{c}}) \geq w^* > t$ $(*)$. This is true, because the minimal weight w^* of $\mathcal{C}$ is equal to d, since $\mathcal{C}$ is linear. In other words: more than t errors have occurred. Suppose that $\tilde{\mathbf{c}}$ is found by bit-flipping t_0 1's and t_1 0's of the bits of $\mathbf{c}$, such that $t_0 + t_1 \leq t$. Then, by construction: $d(\mathbf{c}, \tilde{\mathbf{c}}) = t_0 + t_1 \leq t$ $(**)$. Thus, by $(*)$ and $(**)$, $\tilde{\mathbf{c}}$ is closer to $\mathbf{c}$ than to $\mathbf{0}$, and therefore the decoder returns $\mathbf{c}$. As a consequence, the probability that $\mathbf{0}$ turns into $\tilde{\mathbf{c}}$ is:

$$\begin{aligned} P(\mathbf{0} \text{ turns into } \tilde{\mathbf{c}}) &= p^{w-t_0}(1-p)^{t_0} p^{t_1}(1-p)^{n-w-t_1} \\ &= p^{w-t_0+t_1}(1-p)^{n-w-t_1+t_0} \end{aligned} \tag{2.3}$$

Indeed, $\mathbf{c}$ has weight w and $\tilde{\mathbf{c}}$ is obtained by bit-flipping t_0 (of w) 1's and t_1 (of $n-w$) 0's, thus $\tilde{\mathbf{c}}$ consists of $w - t_0$ 1's that are also 1's in $\mathbf{c}$ and of t_1 1's that are 0's in $\mathbf{c}$ and the remaining bits are 0's. Hence $\tilde{\mathbf{c}}$ has weight $w - t_0 + t_1$, in other words: To obtain $\tilde{\mathbf{c}}$ from $\mathbf{0}$, $w - t_0 + t_1$ bits need to be flipped and the remaining $n - w - t_1 + t_0$ bits remain the same which gives result (2.3). An example has been sketched in Figure 2.5

Now we need to count the amount of code words $\tilde{\mathbf{c}}$ within distance t of $\mathbf{c}$. Notice that we can choose which of the w 1's and which of the $n - w$ 0's to flip in order to get a code word $\tilde{\mathbf{c}}$ within

distance t of $\mathbf{c}$, thus we choose t_0 out of w 1's to flip and t_1 out of $n-w$ 0's to flip. This gives us the following result:

$$P(\mathbf{0} \text{ turns into a code word } \tilde{\mathbf{c}} \text{ within distance } t \text{ of } \mathbf{c}) = \sum_{\substack{t_0,t_1 \geq 0 \\ t_0+t_1 \leq t}} \binom{w}{t_0}\binom{n-w}{t_1} p^{w+t_1-t_0}(1-p)^{n-w-t_1+t_0} \tag{2.4}$$

Notice that $\mathbf{c}$ was an arbitrarily chosen code word of weight w such that an undetected error occurs, the amount of such code words is equal to $A_w^{g_{\text{CRC}}}(\mathcal{C})$, this can then be done for any weight w and this proves the result. □

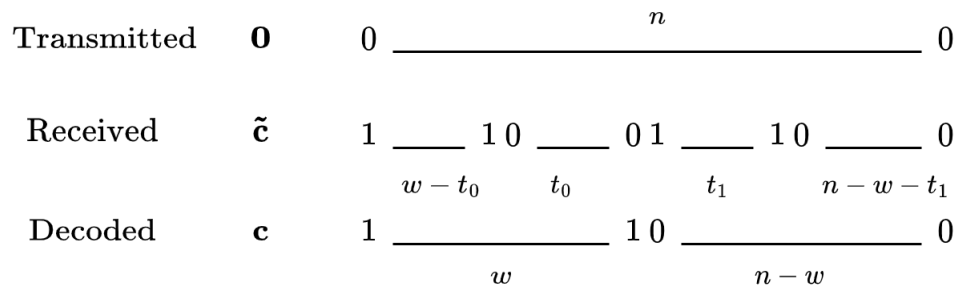

Figure 2.5: An example of the bit-flips that occur in case of an undetected error

Now that we have the formulas for the probability of an undetected error occurring we will give a short overview of them to highlight the difference:

- For the system without an error-correcting code, i.e., the *pure CRC system*, P_{ue} is given by:

$$P_{ue}(p) = \sum_{w=1}^{k} A_w^{g_{\text{CRC}}} p^w (1-p)^{k-w}$$

- For the CRC system in combination with an error-correcting code, i.e. the *coded CRC system*, P_{ue} is given by:

$$P_{ue}(\mathcal{C}, t, p) = \sum_{w=1}^{n} A_w^{g_{\text{CRC}}}(\mathcal{C}) \sum_{\substack{t_0,t_1 \geq 0 \\ t_0+t_1 \leq t}} \binom{w}{t_0}\binom{n-w}{t_1} p^{w+t_1-t_0}(1-p)^{n-w-t_1+t_0}$$

Chapter 3

CRC systems with BCH codes

In this chapter we will divide our evaluation into a few parts. Firstly, in Section 3.1 the definition of a BCH (Bose-Chaudhuri-Hocquenghem) code, which was first introduced in "On a Class of Error Correcting Binary Group" by R.C. Bose and D.K. Ray-Chadhuri [4] is given. Then, in Section 3.2 we will look at a first example of a BCH code. In Section 3.3 we will introduce CRC polynomials that are known in literature to perform quite well and we will see how the BCH codes influences the performance on those polynomials. Then in Section 3.4 we will do a weight analysis in which we will look at the weight distributions of the coded system to see where good performance of CRC polynomials in the coded system comes from. Lastly we will look at a second BCH code in Section 3.5 and do a similar weight analysis on the polynomials.

3.1 Definition BCH Code

The advantage of BCH codes is that we can control the minimal distance (its lower bound) of the code by constructing the generator polynomial in a handy way. This will come in handy later when we want to optimize the performance of the CRC polynomials.

Let n be the length of the binary, cyclic code that we will generate. Let $\delta > 0$ be an integer, which we will call the *designed distance*. Let $\beta, \beta^2, \ldots, \beta^{\delta-1}$ be such that β is a n-th *primitive root of unity*, i.e., β satisfies $\beta^n = 1$. Note that $\beta \in F_{2^m}$, where m is such that n satisfies $n \mid 2^m - 1$. Now, $g_{\text{BCH}}(x)$ is the generator polynomial equal to the product of the *different* minimal polynomials of $\beta, \beta^2, \cdots, \beta^{\delta-1}$. Note that we emphasize the fact that $g_{\text{BCH}}(x)$ only consists of those minimal polynomials that are different, because it is possible that some of them are equal. Equivalently, we have that:

$$g_{\text{BCH}}(x) = \text{LCM}\left(m_\beta(x), m_{\beta^2}(x), \ldots, m_{\beta^{\delta-1}}(x)\right)$$

So we know that for any word w holds that:

$$\begin{aligned} g_{\text{BCH}}(x) \mid w(x) \iff & m_\beta(x) \mid w(x) \text{ and } m_{\beta^2}(x) \mid w(x) \ldots \text{ and } m_{\beta^{\delta-1}}(x) \mid w(x) \\ & w(\beta) = 0 \text{ and } w(\beta^2) = 0 \ldots \text{ and } w(\beta^{\delta-1}) = 0 \end{aligned} \tag{3.1}$$

Note that (3.1) holds, because for any minimal polynomial $m_\beta(x)$, it holds that $m_\beta(\beta) = 0$, thus if $m_\beta(x) \mid w(x)$, this implies that $w(x) = q(x)m_\beta(x)$, for some function q and hence $w(\beta) = q(\beta)m_\beta(\beta) = 0$. Now by (3.1) we note that $w \in \mathcal{C}$ if and only if:

$$wH = 0$$

with H satisfying

$$H = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \beta & \beta^2 & \beta^3 & \cdots & \beta^{\delta-1} \\ \beta^2 & (\beta^2)^2 & (\beta^3)^2 & \cdots & (\beta^{\delta-1})^2 \\ \vdots & \vdots & \vdots & & \vdots \\ \beta^{n-1} & (\beta^2)^{n-1} & (\beta^3)^{n-1} & \cdots & (\beta^{\delta-1})^{n-1} \end{bmatrix} \tag{3.2}$$

which we can rewrite to

$$H = \begin{bmatrix} \beta^0 & (\beta^0)^2 & (\beta^0)^3 & \cdots & (\beta^0)^{\delta-1} \\ \beta & \beta^2 & \beta^3 & \cdots & \beta^{\delta-1} \\ \beta^2 & (\beta^2)^2 & (\beta^2)^3 & \cdots & (\beta^2)^{\delta-1} \\ \vdots & \vdots & \vdots & & \vdots \\ \beta^{n-1} & (\beta^{n-1})^2 & (\beta^{n-1})^3 & \cdots & (\beta^{n-1})^{\delta-1} \end{bmatrix} \tag{3.3}$$

Now notice that we can take any $\delta - 1$ pair of rows to get a matrix A of the form

$$A = \begin{bmatrix} a_1 & (a_1)^2 & (a_1)^3 & \cdots & (a_1)^{\delta-1} \\ a_2 & (a_2)^2 & (a_2)^3 & \cdots & (a_3)^{\delta-1} \\ a_3 & (a_3)^2 & (a_3)^3 & \cdots & (a_3)^{\delta-1} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{\delta-1} & (a_{\delta-1})^2 & (a_{\delta-1})^3 & \cdots & (a_{\delta-1})^{\delta-1} \end{bmatrix} \tag{3.4}$$

Where $a_i \in \{\beta^0, \beta, \beta^2, \ldots, \beta^{n-1}\}$ for all $i = 1, \ldots, \delta - 1$ and $a_i \neq a_j$, for $i \neq j$. This is a square matrix so we can calculate its determinant to find

$$\det(A) = a_1 a_2 \cdots a_{\delta-1} \begin{vmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{\delta-2} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{\delta-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_{\delta-1} & a_{\delta-1}^2 & \cdots & a_{\delta-1}^{\delta-2} \end{vmatrix} \tag{3.5}$$

$$= a_1 a_2 \cdots a_{\delta-1} \prod_{1 \leq j \leq i \leq (\delta-1)} (a_i - a_j) \tag{3.6}$$

$$\neq 0 \tag{3.7}$$

Note that the determinant in (3.5) is a Vandermonde matrix, which gives (3.6) (proof in [5]). Hence each pair of $\delta - 1$ rows is independent and we can conclude that $d > \delta - 1$ or in other words $d \geq \delta$.

3.2 [31,21,5] BCH code

In [3] a great example is given where a collection of CRC polynomials are compared in the pure system and in the coded system. The peculiar result is that a CRC polynomial that performs well in the pure system, might not perform as good in the coded system. In the article the code used is a BCH code with $n = 31$, the dimension, $k = 21$ and the designed distance $\delta = 5$. Thus we get a BCH code that has minimal distance $d \geq \delta = 5$ that is generated by a generator polynomial consisting of the distinct minimal polynomials of β, β^2, β^3 and β^4, where $\beta^{31} = 1$ and β is a primitve n-th unity root from F_{32}. Using the method described in Section 3.1 we get the following generator polynomial:

$$g_{\text{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1 \tag{3.8}$$

Note that $0 \leq t \leq \lfloor \frac{d-1}{2} \rfloor$, thus we can use a 2-bounded-distance decoder. By generating all the code words we find that there exists a code word of weight $w = 5$ and therefore $w^* = 5$. Note that if $c(x) = x^{20} + x^{19} + x^{17}$, then $g_{\text{BCH}}(x)c(x) = x^{17} + x^{19} + x^{24} + x^{25} + x^{30}$. The word $c(x)$ is a message word of 21 bits and $g_{\text{BCH}}(x)c(x)$ is a BCH code word of weight 5, so indeed $w^* = 5$. This BCH code is used in systems such as in paging protocols [7]. In [3] an example is illustrated of 6-bits CRC polynomials, i.e., CRC polynomials that add 6 bits to message words. Here the polynomials $g_{\text{CRC}_1}(x) = x^6 + x^2 + x + 1$, $g_{\text{CRC}_2}(x) = x^6 + x^5 + x^2 + x + 1$, $g_{\text{CRC}_3}(x) = x^6 + x^4 + x^3 + 1$ and $g_{\text{CRC}_4}(x) = x^6 + x^5 + x^4 + x^3 + x + 1$ are defined. In the pure system $w^*_{g_{\text{CRC}_1}} = w^*_{g_{\text{CRC}_3}} = w^*_{g_{\text{CRC}_4}} = 4$ and $w^*_{g_{\text{CRC}_2}} = 3$. The article shows that $A_w^{g_{\text{CRC}_1}} = A_w^{g_{\text{CRC}_4}}$ for all w, so that only $g_{\text{CRC}_3}(x)$ and $g_{\text{CRC}_4}(x)$ have to be compared, since $g_{\text{CRC}_2}(x)$ performs worse due to a lower minimal weight . g_{CRC_3} performs optimally in the pure system since $A_{w^*}^{g_{\text{CRC}_1}} = 204 < 205 = A_w^{g_{\text{CRC}_4}}$. Here $g_{\text{CRC}_3}(x)$ gives only a slight improvement. However in the coded system the minimal weights are $w^*_{g_{\text{CRC}_1}} = w^*_{g_{\text{CRC}_2}} = 6$ and $w^*_{g_{\text{CRC}_3}} = w^*_{g_{\text{CRC}_4}} = 7$, with $A_{w^*}^{g_{\text{CRC}_4}}(\mathcal{C}) = 310 < 465 = A_{w^*}^{g_{\text{CRC}_3}}(\mathcal{C})$, so $g_{\text{CRC}_4}(x)$ is optimal in the coded system. The article showed that the respective performance of the CRC polynomials is different when comparing the results in the pure system to those in the coded system. However, this is only one example and $g_{\text{CRC}_3}(x)$ performs only slightly better than $g_{\text{CRC}_4}(x)$ in the pure system.

We want to find other CRC polynomials that show the same behaviour of CRC polynomials that perform well in the pure system, but perform badly in the coded system, or the other way around. In the next section we will give such an example.

3.3 CRC polynomials of degree 5

In this example we will examine the performance of CRC polynomials in both CRC systems, i.e., the pure system and the coded sytem, for CRC polynomials of degree 5. These are CRC polynomials that append 5 bits to the data. Since we use a $[31, 21, 5]$ BCH code and CRC polynomials of degree 5, the CRC polynomials must append to 16 bits of data, to get message words of length $k = 5 + 16 = 21$ that can be used to be encoded into a BCH code word of length $n = 31$. As discussed by Koopman in [8] a variety of choices of CRC polynomials can be made. We will look at the polynomials that are denoted by **0x15**, **0x12**, **0x1e** in the Koopman notation [9]. In Table 3.1 we give an overview of these polynomials and their corresponding notation and the name that these polynomials have been given in literature. Note that their are often multiple names given to certain polynomials, here the names are used that are given in [8].

Notation	CRC Polynomial	Name
0x15	$g_{\text{CRC}_1}(x) = x^5 + x^3 + x + 1$	CRC-5, (CCITT-5 [10])
0x12	$g_{\text{CRC}_2}(x) = x^5 + x^2 + 1$	CRC-USB-5
0x1e	$g_{\text{CRC}_3}(x) = x^5 + x^4 + x^3 + x^2 + 1$	CRC-5F/3

Table 3.1: The polynomial representation of the CRC polynomials CRC-5, CRC-USB-5 and CRC-5F/3 together with their (Koopman) notation

In [8] it is discussed that the minimal distance of each of these CRC codes (in the pure system) that are generated by the corresponding CRC polynomials varies by the amount of data bits the CRC polynomials append to and by the polynomial that is used. CRC-USB-5 is a polynomial that is mainly used in USB systems, to protect USB token packets [11]. According to [8], CRC-5 has the largest corresponding minimal distance (when appending to data of up to 10 bits) and thus usually this polynomial is chosen as optimal, since a higher minimum weight contributes less to

P_{ue} than lower weights for small values of p. However we do note that our data consists of 16 bits and therefore the minimal weight of the code generated by CRC-5 is lower and we predict that the performance of this polynomials will be less optimal.

3.3.1 Undected error probability in the pure system

The minimum weights of the codes corresponding to the CRC polynomials in the pure system are given by $w^*_{g_{\text{CRC}_1}} = 2$, $w^*_{g_{\text{CRC}_2}} = 3$ and $w^*_{g_{\text{CRC}_3}} = 3$. Therefore either $g_{\text{CRC}_2}(x)$ or $g_{\text{CRC}_3}(x)$ performs optimally. In Appendix A the weight distributions of these three CRC polynomials is given. Notice that $A^{g_{\text{CRC}_2}}_{w^*} = 45 < 47 = A^{g_{\text{CRC}_3}}_{w^*}$, so $g_{\text{CRC}_2}(x)$ performs slightly better than $g_{\text{CRC}_3}(x)$. In Figure 3.1 the undetected error probability P_{ue} is plotted against p for the three different CRC polynomials. As can be seen, $g_{\text{CRC}_2}(x)$ indeed seems to perform the best and $g_{\text{CRC}_1}(x)$ is performing the worst. Thus, as predicted, in the pure system when using these parameters for the data (16 bits of data), CRC-5 does not perform well in the pure system and CRC-USB-5 performs the best.

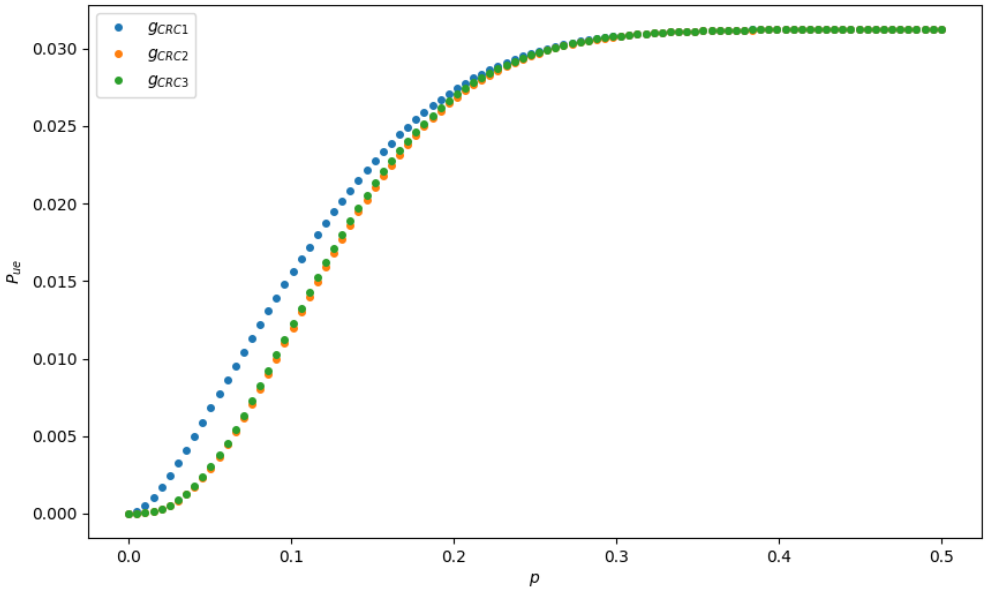

Figure 3.1: $P_{ue}(p)$ for the three different CRC polynomials using the pure system

3.3.2 Undetected error probability for the coded system

Now if we combine these CRC polynomials with the BCH code generated by $g_{\text{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$, the weight distribution changes. The minimum weights are given by $w^*_{g_{\text{CRC}_1}} = 6$, $w^*_{g_{\text{CRC}_2}} = 5$ and $w^*_{g_{\text{CRC}_3}} = 5$. Thus the minimum weight of CRC-5 is now the highest of the three. Therefore CRC-5, which is the only polynomial of these three with minimal weight equal to 6, will perform optimally. Also note that $A^{g_{\text{CRC}_2}}_{w^*}(\mathcal{C}) = A^{g_{\text{CRC}_3}}_{w^*}(\mathcal{C}) = 6$ and $A^{g_{\text{CRC}_2}}_{3}(\mathcal{C}) = 18 < 27 = A^{g_{\text{CRC}_3}}_{4}(\mathcal{C})$, which can be seen in Appendix A. So $g_{\text{CRC}_4}(x)$ performs the worst of these three in the coded system. For the undetected error probability we get the following results that are shown in Figure 3.2. Notice that, indeed CRC-5 performs the best out of the three polynomials and the best performing polynomial in the pure system, CRC-USB-5, is now performing worse in the coded system. Thus the best option when choosing a CRC polynomial that appends 5 bits to 16 bits of

data, is CRC-USB-5 in the pure system and CRC-5 in the coded system when using the BCH code generated by $g_{\text{BCH}}(x)$.

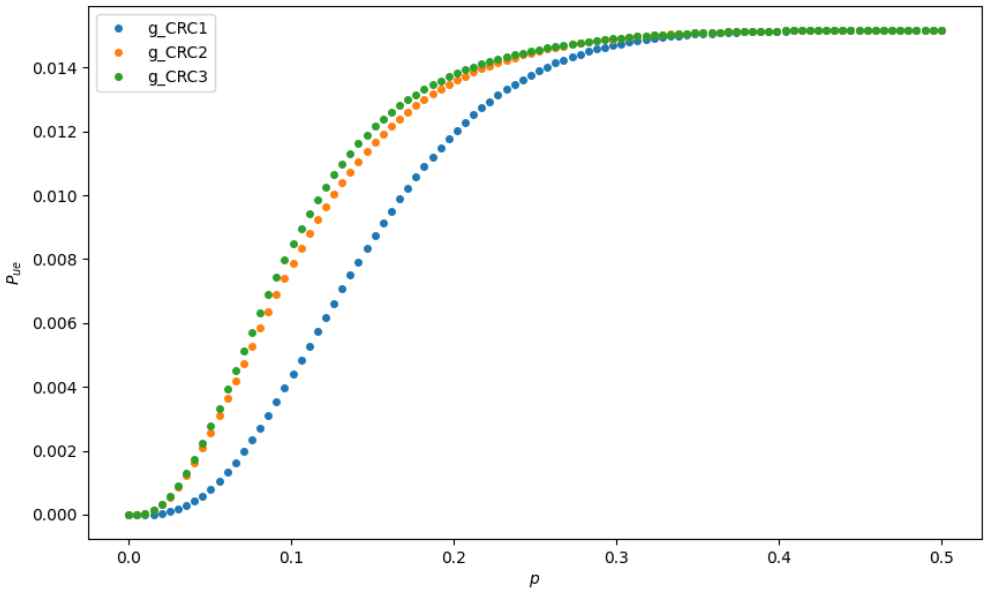

Figure 3.2: $P_{ue}(C, 2, p)$ for the three different CRC polynomials using the coded system

One way to see why these weights have improved is to look at the roots of the polynomials $g_{\text{CRC}_1}(x)g_{\text{BCH}}(x)$, $g_{\text{CRC}_2}(x)g_{\text{BCH}}(x)$ and $g_{\text{CRC}_3}(x)g_{\text{BCH}}(x)$, which will be explained in the next section.

3.4 Weight Analysis

The first obvious thing to do is to see to where the CRC words map to in the resulting BCH code. For each CRC code we can take the words of a fixed weight and look at the words they produce in the coded system, to see whether there is a pattern visible. Unfortunately there does not seem to be a very clear pattern in the code words. The results are shown in Appendix B.

For this weight analysis we will first note a few things. In the coded scheme any message word $\mathbf{a}$ is first converted into a CRC word $\mathbf{c}$ by multiplying with a CRC polynomial $g_{\text{CRC}}(x)$, whereafter the CRC word is converted into a BCH word $\mathbf{u}$ using the BCH generator polynomial $g_{\text{BCH}}(x)$. Equivalently, we can directly convert $\mathbf{a}$ into $\mathbf{u}$ by multiplying with the product of the two generator polynomials $g_{\text{CRC}}(x)g_{\text{BCH}}(x)$. Thus when encoding we can multiply any message word $\mathbf{a}$ with $g(x)$ to get the same cyclic code. This gives rise to analyse the generator polynomial $g_{\text{CRC}}(x)g_{\text{BCH}}(x)$. For the three chosen CRC polynomials in the previous section we get the following polynomials:

$$g_1(x) = g_{\text{CRC}_1}(x)g_{\text{BCH}}(x) = x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^4 + x + 1$$
$$g_1(x) = g_{\text{CRC}_2}(x)g_{\text{BCH}}(x) = x^{15} + x^{14} + x^{13} + x^{12} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + 1$$
$$g_1(x) = g_{\text{CRC}_3}(x)g_{\text{BCH}}(x) = x^{15} + x^{13} + x^{12} + x^{11} + x^9 + x^5 + x^4 + x^2 + 1$$

Now, as noted in Section 3.2, by construction β, β^2, β^3 and β^4 are roots from F_{32}. Where β is a primitive n-th unity root. We can write $F_{32} = \{0, 1, \beta, \beta^2, \ldots, \beta^{30}\} = F_2[x]/(f(x))$, where $f(x)$ is an irreducible polynomial of degree 5. Any irreducible polynomial of degree 5 can be chosen but in this example we have chosen for $x^5 + x^2 + 1$. The table for F_{32} [12]can be found in Appendix C.

Now, using Table C.1 we can calculate $g(\beta^i)$ for $i = 0, \ldots, 30$ for a generator polynomial $g(x)$. If $g(\beta^i) = 0$, then β^i is a root of $g(x)$. We do this for the generator polynomials $g_1(x), g_2(x)$ and $g_3(x)$. The results are shown in Table 3.2. For comparison we have also shown the roots of $g_{\text{BCH}}(x)$ and of $g_{\text{CRC}_1}(x)$, $g_{\text{CRC}_2}(x)$ and $g_{\text{CRC}_3}(x)$.

Polynomial	**Roots from $\mathbf{F_{32}}$**
$g_1(x)$	$1, \beta, \beta^2, \beta^3, \beta^4, \beta^6, \beta^8, \beta^{12}, \beta^{16}, \beta^{17}, \beta^{24}$
$g_2(x)$	$\beta, \beta^2, \beta^3, \beta^4, \beta^6, \beta^8, \beta^{12}, \beta^{16}, \beta^{17}, \beta^{24}$
$g_3(x)$	$\beta, \beta^2, \beta^3, \beta^4, \beta^6, \beta^8, \beta^{12}, \beta^{16}, \beta^{17}, \beta^{24}$
$g_{\text{BCH}}(x)$	$\beta, \beta^2, \beta^3, \beta^4, \beta^6, \beta^8, \beta^{12}, \beta^{16}, \beta^{17}, \beta^{24}$
$g_{\text{CRC}_1}(x)$	1
$g_{\text{CRC}_2}(x)$	$\beta^1, \beta^2, \beta^4, \beta^8, \beta^{16}$
$g_{\text{CRC}_3}(x)$	$\beta^3, \beta^6, \beta^{12}, \beta^{17}, \beta^{24}$

Table 3.2: The roots from F_{32} for each generator polynomial. For $g_1(x), g_2(x)$, $g_3(x)$ and $g_{\text{BCH}}(x)$ the consecutive roots have been marked

The peculiar thing that can be seen from Table 3.2 is that $g_1(x)$ has the same roots as $g_2(x)$, $g_3(x)$ and $g_{\text{BCH}}(x)$, except $g_1(x)$ also has 1 as root. This becomes immediately clear when you notice that $g_{CRC}(x)$ also has 1 as root, which is a root that $g_{\text{BCH}}(x)$ does not have. Also notice that with the extra root 1, $g_1(x)$ has now 5 consecutive powers of β as roots, whereas $g_{\text{BCH}}(x)$ has only 4. Also notice that the other CRC polynomials do not have any roots different from $g_{\text{BCH}}(x)$, this is the deciding factor for the minimum weights of the coded CRC systems. The following theorem is taken from [3] and the proof is basically the same as the proof that any BCH code with designed distance δ has minimal distance $d \geq \delta$, taken from [13].

Theorem 3.1. *Let $g(x)$ be a generator polynomial of a cyclic code $\mathcal{C}$, with length n. Suppose that $g(x)$ has $\delta - 1$ roots that are consecutive powers of β, i.e. $\beta^b, \beta^{b+1}, \cdots, \beta^{b+\delta-2}$. Then the minimum weight w^* of the corresponding cyclic code satisfies $w^* \geq \delta$.*

Proof. We start by noting that $\beta^b, \beta^{b+1}, \cdots, \beta^{b+\delta-2}$ are roots of $g(x)$. So we know that:

$$\text{LCM}\left(m_{\beta^b}(x), m_{\beta^{b+1}}(x), \ldots, m_{\beta^{b+\delta-2}}(x)\right) \mid g(x) \tag{3.9}$$

Thus for any word w of length n it holds that $w \in \mathcal{C} \iff w(\beta^b) = 0$ and $w(\beta^{b+1}) = 0 \ldots$ and $w(\beta^{b+\delta-2}) = 0$, or equivalently:

$$w \in \mathcal{C} \iff wH = 0$$

with

$$H = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \beta^b & \beta^{b+1} & \beta^{b+2} & \cdots & \beta^{b+\delta-2} \\ (\beta^b)^2 & (\beta^{b+1})^2 & (\beta^{b+2})^2 & \cdots & (\beta^{b+\delta-2})^2 \\ \vdots & \vdots & \vdots & & \vdots \\ (\beta^b)^{n-1} & (\beta^{b+1})^{n-1} & (\beta^{b+2})^{n-1} & \cdots & (\beta^{b+\delta-2})^{n-1} \end{bmatrix} \tag{3.10}$$

which we can rewrite to:

$$H = \begin{bmatrix} (\beta^0)^b & (\beta^0)^{b+1} & (\beta^0)^{b+2} & \cdots & (\beta^0)^{b+\delta-2} \\ \beta^b & \beta^{b+1} & \beta^{b+2} & \cdots & \beta^{b+\delta-2} \\ (\beta^2)^b & (\beta^2)^{b+1} & (\beta^2)^{b+2} & \cdots & (\beta^2)^{b+\delta-2} \\ \vdots & \vdots & \vdots & & \vdots \\ (\beta^{n-1})^b & (\beta^{n-1})^{b+1} & (\beta^{n-1})^{b+2} & \cdots & (\beta^{n-1})^{b+\delta-2} \end{bmatrix} \tag{3.11}$$

Now notice that we can take any $\delta - 1$ pair of rows to get a matrix A of the form

$$A = \begin{bmatrix} a_1^b & a_1^{b+1} & a_1^{b+2} & \cdots & a_1^{b+\delta-2} \\ (a_2^2)^b & (a_2^2)^{b+1} & (a_2^2)^{b+2} & \cdots & (a_2^2)^{b+\delta-2} \\ \vdots & \vdots & \vdots & & \vdots \\ (a_{\delta-1}^{\delta-1})^b & (a_{\delta-1}^{\delta-1})^{b+1} & (a_{\delta-1}^{\delta-1})^{b+2} & \cdots & (a_{\delta-1}^{\delta-1})^{b+\delta-2} \end{bmatrix} \tag{3.12}$$

Where $a_i \in \{\beta^0, \beta^1, \beta^2, \ldots, \beta^{n-1}\}$ for all $i = 1, \ldots, \delta - 1$ and $a_i \neq a_j$, for $i \neq j$. This is a square matrix so we can calculate its determinant to find

$$\det(A) = (a_1^b a_2^b \cdots a_{\delta-1}^b) \begin{vmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{\delta-2} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{\delta-2} \\ 1 & a_3 & a_3^2 & \cdots & a_3^{\delta-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_{\delta-1} & a_{\delta-1}^2 & \cdots & a_{\delta-1}^{\delta-2} \end{vmatrix} \tag{3.13}$$

Notice that the determinant in (3.13) is exactly the same as the determinant in (3.5), and thus we conclude that $\det(A) \neq 0$. Since we had chosen an arbitrary subset of rows, we conclude that all pairs of $\delta - 1$ rows of H are independent and therefore minimal distance d of $\mathcal{C}$ satisfies $d \geq \delta$ and since $d = w^*$, the result is proven. □

Now we can immediately conclude by Theorem 3.1 that $w^*_{\mathrm{CRC}_1} \geq 6, w^*_{\mathrm{CRC}_2} \geq 5$ and $w^*_{\mathrm{CRC}_3} \geq 5$ in the coded system. Note that the fact that $g_{\mathrm{CRC}_1}(x)$ has an even amount of terms, is the determining factor that makes the lower bound for the minimum weight better compared to the other CRC polynomials. This is indeed the case since an even amount of terms immediately implies that $g_{\mathrm{CRC}_1}(1) = 0$, and thus also $g_1(1) = 0$. The other two polynomials have an odd amount of terms, wherefore these polynomials have one less consecutive root compared to $g_{\mathrm{CRC}_1}(1)$ which explains the lower bound. Thus when combining with the BCH code, the best strategy is to choose a CRC polynomial that has the most amount of roots different to the roots of the BCH code such that they form a larger set of consecutive roots. Thus, to optimize we can look at all the different CRC polynomials of 5 bits and examine their roots. In our case we need to find CRC polynomials that have roots consecutive to β, β^2, β^3 and β^4. Therefore we will look at the roots (from F_{32}) of all the possible CRC polynomials of 5 bits, to find the one having the best possible lower bound on the minimal weight. These roots are found by plugging all powers of β in the generator polynomial and if it equals zero, that power of beta is a root. Below I have made a list of CRC polynomials and the powers of beta that are roots of the CRC polynomials .

Polynomial	**Definition**	**Roots from $\mathbf{F_{32}}$**	$w^* \geq$
$g_{\mathrm{CRC}_1}(x)$	x^5+x^3+x+1	1	6
$g_{\mathrm{CRC}_2}(x)$	x^5+x^2+1	$\beta^1, \beta^2, \beta^4, \beta^8, \beta^{16}$	5
$g_{\mathrm{CRC}_3}(x)$	$x^5+x^4+x^3+x^2+1$	$\beta^3, \beta^6, \beta^{12}, \beta^{17}, \beta^{24}$	5
$g_{\mathrm{CRC}_4}(x)$	x^5+x+1	None	5
$g_{\mathrm{CRC}_5}(x)$	$x^5+x^4+x^3+1$	1	6
$g_{\mathrm{CRC}_6}(x)$	$x^5+x^3+x^2+1$	1	6
$g_{\mathrm{CRC}_7}(x)$	x^5+x^4+x+1	1	6
$g_{\mathrm{CRC}_8}(x)$	$x^5+x^4+x^3+x^2+x+1$	1	6
$g_{\mathrm{CRC}_9}(x)$	x^5+x^2+x+1	1	6
$g_{\mathrm{CRC}_{10}}(x)$	$x^5+x^4+x^2+1$	1	6
$g_{\mathrm{CRC}_{11}}(x)$	$x^5+x^3+x^2+x+1$	$\beta^7, \beta^{14}, \beta^{19}, \beta^{25}, \beta^{28}$	5
$g_{\mathrm{CRC}_{12}}(x)$	$x^5+x^4+x^2+x+1$	$\beta^5, \beta^9, \beta^{10}, \beta^{18}, \beta^{20}$	7
$g_{\mathrm{CRC}_{13}}(x)$	x^5+x^3+1	$\beta^{15}, \beta^{23}, \beta^{27}, \beta^{29}, \beta^{30}$	5
$g_{\mathrm{CRC}_{14}}(x)$	x^5+x^4+1	None	5
$g_{\mathrm{CRC}_{15}}(x)$	x^5+1	1	6
$g_{\mathrm{CRC}_{16}}(x)$	$x^5+x^4+x^3+x+1$	$\beta^{11}, \beta^{13}, \beta^{21}, \beta^{22}, \beta^{26}$	5

Table 3.3: The roots from F_{32} for each CRC polynomial and a lower bound for the minimal weight in the coded system with $g_{\mathrm{BCH}}(x)$ according to Theorem 3.1

As can be seen from Table 3.3 $g_{\mathrm{CRC}_{12}}(x)$ has the best lower bound on the minimal weight in the coded CRC scheme. If we look at the weight distribution, we see that indeed $g_{\mathrm{CRC}_{12}}(x)$ has the highest minimal weight with $w^*_{\mathrm{CRC}_{12}} = 7$. See also Table 3.4, where the lower bound is compared to the actual value of w^*. Notice that $g_{\mathrm{CRC}_{16}}(x)$ also has minimal weight $w^* = 7$. However $g_{\mathrm{CRC}_{16}}(x)$ has exactly the same weight distribution as $g_{\mathrm{CRC}_{12}}(x)$ (see Appendix A), both in the pure and in the coded system and will thus generate the same results in terms of undetected error as $g_{\mathrm{CRC}_{12}}(x)$. Thus one can interchange both polynomials.

Polynomial	**Lower bound on w^***	**Actual w^***
$g_{\mathrm{CRC}_1}(x)$	6	6
$g_{\mathrm{CRC}_2}(x)$	5	5
$g_{\mathrm{CRC}_3}(x)$	5	5
$g_{\mathrm{CRC}_4}(x)$	5	6
$g_{\mathrm{CRC}_5}(x)$	6	6
$g_{\mathrm{CRC}_6}(x)$	6	6
$g_{\mathrm{CRC}_7}(x)$	6	6
$g_{\mathrm{CRC}_8}(x)$	6	6
$g_{\mathrm{CRC}_9}(x)$	6	6
$g_{\mathrm{CRC}_{10}}(x)$	6	6
$g_{\mathrm{CRC}_{11}}(x)$	5	6
$g_{\mathrm{CRC}_{12}}(x)$	7	7
$g_{\mathrm{CRC}_{13}}(x)$	5	5
$g_{\mathrm{CRC}_{14}}(x)$	5	5
$g_{\mathrm{CRC}_{15}}(x)$	6	5
$g_{\mathrm{CRC}_{16}}(x)$	5	7

Table 3.4: A lower bound on w^* provided by Theorem 3.1 and the actual value of w^*

We also note that $g_{\mathrm{CRC}_{12}}(x)$ has $w^*_{\mathrm{CRC}_{12}} = 3$ in the pure system and therefore will probably have similar performance compared to $g_{\mathrm{CRC}_1}(x)$, which can also be seen in Appendix A. Another thing that can be noticed from Table 3.3 is that the square of each root is again a root for each of the polynomials. This is no coincidence, which will be shown in the following lemma:

Lemma 3.1.1. *If β is a root of a polynomial $g(x)$, then β^2 is also a root.*

Proof. Suppose that β is a root of $g(x)$. Suppose that $g(x)$ has degree k. Write

$$g(x) = a_0 + a_1 x + a_2 x^2 + a_3 + x^3 \cdots + a_{k-1} x^{k-1}$$

Then:

$$\begin{aligned} g(\beta^2) &= a_0 + a_1\beta^2 + a_2(\beta^2)^2 + a_3(\beta^2)^3 + \cdots + a_{k-1}(\beta^2)^{k-1} \\ &= a_0^2 + a_1^2\beta^2 + a_2^2(\beta^2)^2 + a_3^2(\beta^3)^2 + \cdots + a_{k-1}^2(\beta^{k-1})^2 \\ &= a_0^2 + (a_1\beta)^2 + (a_2\beta^2)^2 + (a_3\beta^3)^2 + \cdots + (a_{k-1}\beta^{k-1})^2 \end{aligned}$$

Where we used that $a^p = a \pmod p$ for any prime number p, by Fermat's Little Theorem, with $p = 2$. Now note that $(a+b)^2 = a^2 + 2ab + b^2 = a^2 + b^2$ for any a, b when calculating binary. By induction it can easily be proven that $(a_0 + a_1 + \cdots + a_{n-1})^2 = a_0^2 + a_1^2 + \cdots + a_{n-1}^2$ for any n, when calculating binary. Therefore:

$$\begin{aligned} g(\beta^2) &= (a_0 + a_1\beta^1 + a_2\beta^2 + a_3\beta^3 + \cdots + a_{k-1}\beta^{k-1})^2 \\ &= (g(\beta))^2 = 0 \end{aligned}$$

Which proves the result. □

This is a result that we will use later when we introduce a different BCH code.

Now that we have found that $g_{\mathrm{CRC}_{12}}(x)$ has the highest minimal weight, we can compare the results of $g_{\mathrm{CRC}_{12}}(x)$ to those of $g_{\mathrm{CRC}_1}(x), g_{\mathrm{CRC}_2}(x)$ and $g_{\mathrm{CRC}_3}(x)$ with respect to the undetected error both in the pure and in the coded system. The results are shown in Figure 3.3.

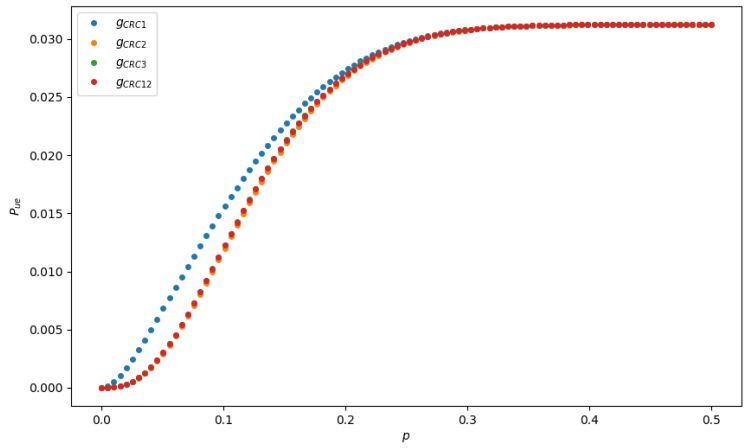

Figure 3.3: The undetected error P_{ue} against the channel error rate p for $g_{\mathrm{CRC}_1}(x), g_{\mathrm{CRC}_2}(x)$ and $g_{\mathrm{CRC}_3}(x)$ in the pure system.

Notice that in the pure system $g_{\mathrm{CRC}_{12}}(x)$ does not necessarily improve on the other three CRC polynomials. However if we look at the curve of the undetected error in the coded system we get different results, which is expected since the minimal weight produced by $g_{\mathrm{CRC}_{12}}(x)$ is higher than the other three. The results are shown in Figure 3.4.

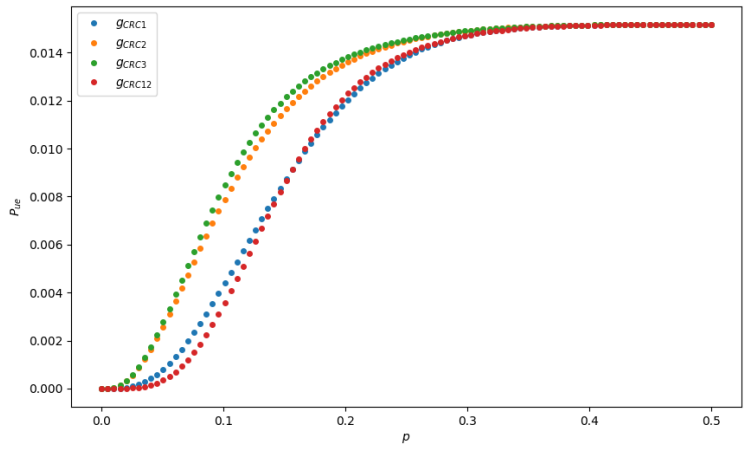

Figure 3.4: The undetected error P_{ue} against the channel error rate p for $g_{\mathrm{CRC}_1}(x), g_{\mathrm{CRC}_2}(x)$ and $g_{\mathrm{CRC}_3}(x)$ in the coded system.

As can be seen from Figure 3.4 $g_{\mathrm{CRC}_{12}}(x)$ does seem to improve on the polynomial $g_{\mathrm{CRC}_1}(x)$, but only for values of p between 0.0 and approximately 0.16. Finally to make sure there are no better performing CRC polynomials we will compare all 16 CRC polynomials to each other. The results for the pure system can be seen in Figure 3.5. Note that the plot for some of the CRC polynomials is not visible, this is due to the fact that for all w holds that $A_w^{g_{\mathrm{CRC}_1}} = A_w^{g_{\mathrm{CRC}_{10}}}$, $A_w^{g_{\mathrm{CRC}_2}} = A_w^{g_{\mathrm{CRC}_{13}}}$, $A_w^{g_{\mathrm{CRC}_3}} = A_w^{g_{\mathrm{CRC}_{11}}}$, $A_w^{g_{\mathrm{CRC}_4}} = A_w^{g_{\mathrm{CRC}_{14}}}$, $A_w^{g_{\mathrm{CRC}_5}} = A_w^{g_{\mathrm{CRC}_9}}$, $A_w^{g_{\mathrm{CRC}_{12}}} = A_w^{g_{\mathrm{CRC}_{16}}}$. Therefore there are only 9 distinct weight distributions for these CRC polynomials, so the plot of 7 of these CRC polynomials is not visibile, but that is no problem since one can look at the plot of their respective equivalent CRC polynomial, i.e., the one that generates the same weight distributions. One thing that can be seen from Figure 3.5 is that the plot of some of these CRC polynomials is not increasing. We note that for $p = \frac{1}{2}$, $P_{ue} = \sum_{w=1}^{k} A_w^{g_{\mathrm{CRC}}}(\frac{1}{2})^k$. Also note that $\sum_{w=1}^{k} A_w^{g_{\mathrm{CRC}}} = 2^m - 1$, where m is the amount of data bits that the CRC polynomial appends to. So $P_{ue}(\frac{1}{2}) = \frac{2^m-1}{2^k}$. So for $p = \frac{1}{2}$, m data bits, the undetected error probability is the same for any CRC polynomial of degree $l = k - m$. Therefore, as can also be seen from Figure 3.4, if $p \to \frac{1}{2}$, $P_{ue} \to \frac{2^{16}-1}{2^{21}} \approx 0.031$. Note that $\lim_{p \to \frac{1}{2}} P_{ue}(p) = P_{ue}(\frac{1}{2})$, because $P_{ue}(p)$ is a continuous function. This at least gives an explanation to the fact that for some of these plots the value for P_{ue} decreases whenever $P_{ue} > \frac{2^{16}-1}{2^{21}}$, for some $0 < p < \frac{1}{2}$.

If you look closely, it can be seen that $g_{\mathrm{CRC}_{14}}(x)$ performs the best for the pure system, together with $g_{\mathrm{CRC}_4}(x)$, because they have the same weight distribution in the pure system as can be seen in Appendix A. This is due to the fact that $g_{\mathrm{CRC}_{14}}(x)$ has the highest minimal weight $w^* = 3$ and the least amount of code words of minimal weight ($A_3^{g_{\mathrm{CRC}_{14}}} = 42$). Therefore it is advised to choose either $g_{\mathrm{CRC}_{14}}(x)$ or $g_{\mathrm{CRC}_4}(x)$ when using the pure system.

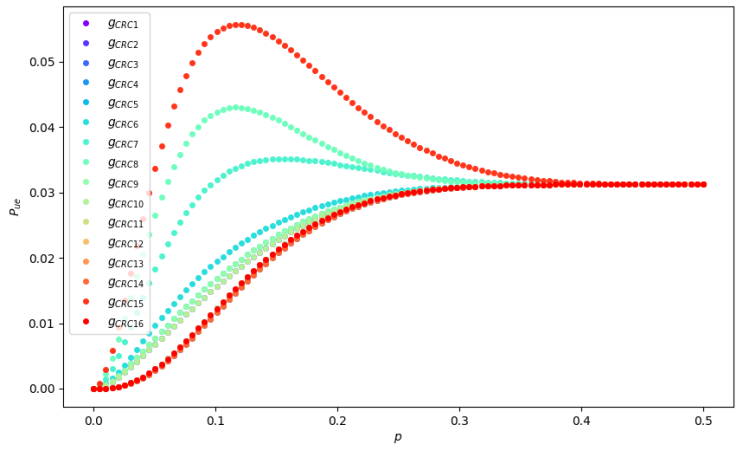

Figure 3.5: The undetected error P_{ue} against the channel error rate p for the $g_{\mathrm{CRC}_1}(x), \ldots, g_{\mathrm{CRC}_{16}}(x)$ CRC polynomials in the pure system.

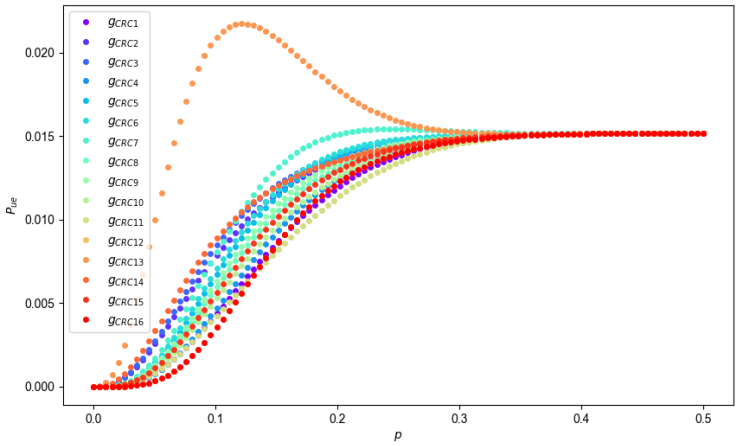

Figure 3.6: The undetected error P_{ue} against the channel error rate p for the $g_{\mathrm{CRC}_1}(x), \ldots, g_{\mathrm{CRC}_{16}}(x)$ CRC polynomials in the coded system.

In Figure 3.6 the performance of $g_{\mathrm{CRC}_1}(x), \ldots g_{\mathrm{CRC}_{16}}(x)$ is plotted together for the coded system. Notice that $g_{\mathrm{CRC}_{16}}(x)$ performs the best, and therefore also $g_{\mathrm{CRC}_{12}}(x)$ since they have the same weight distribution in both systems, however only for small values of p. For larger values (starting from somewhere between 0.1 and 0.2) of p, $g_{\mathrm{CRC}_{12}}(x)$ seems to perform better. Therefore it is advised to choose $g_{\mathrm{CRC}_{16}}(x)$ or $g_{\mathrm{CRC}_{11}}(x)$ for small values of p and $g_{\mathrm{CRC}_{11}}(x)$ for larger values of p when using the coded system. Thus, indeed with conditions on the value for p we can conclude that Theorem 3.1 can provide a method in finding the optimal CRC polynomial for the coded system. Also note that in the coded system $P_{ue}(p)$ decreases for p high enough, for some CRC polynomials. In the coded system $P_{ue}(\mathcal{C}, t, \frac{1}{2}) = \frac{2^m-1}{2^n} \sum_{w=0}^{t} \binom{n}{w}$, as found in [3], for each CRC polynomial that appends $l = k - m$ to m bits of data in combination with an $[n, k, d]$ code. So

$\lim_{p\to\frac{1}{2}} P_{ue}(\mathcal{C},t,p) = \frac{2^m-1}{2^n}\sum_{w=0}^{t}\binom{n}{w}$, which explains why $P_{ue}(\mathcal{C},t,p)$ has to decrease for p high enough, whenever $P_{ue}(\mathcal{C},t,p) > \frac{2^m-1}{2^n}\sum_{w=0}^{t}\binom{n}{w}$ for some $0 < p < \frac{1}{2}$.

3.5 [31,16,7] BCH code

In this section we will look at a second BCH code, a [31,16,7] BCH code, to see whether the results also apply to other codes. In theory we could look at any cyclic error-correcting code but the handy property of BCH codes is that they are constructed by taking the minimal polynomials of consecutive roots. In this section we will look at a BCH code with length $n = 31$, dimension $k = 16$ and designed distance $\delta = 7$. This implies that we have a generator polynomial that is equal to the product of all the distinct minimal polynomials of the $\beta, \beta^2, \beta^3, \beta^4, \beta^5$ and β^6, where $\beta^{31} = 1$, and β is a primitive n-th unity root $\in F_{32}$. Using the method described in Section 3.1 and [12]we get the following generator polynomial:

$$g_{\mathrm{BCH}_2}(x) = x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1$$

Notice that $\delta = 7$ and thus $d \geq \delta = 7$ hence $w^* \geq 7$. Also, since $0 \leq t \leq \lfloor\frac{d-1}{2}\rfloor$, we can use a 3-bounded distance decoder.

If we let β to be the primitive element of $F_{32} \cong F_2[x]/(x^5+x^2+1)$, i.e., β satisfies $\beta^5 = \beta^2+1$, we find that $\beta, \beta^2, \beta^3, \beta^4, \beta^5, \beta^6, \beta^8, \beta^9, \beta^{10}, \beta^{12}, \beta^{13}, \beta^{16}, \beta^{17}, \beta^{18}, \beta^{20}$ and β^{24} are the roots from F_{32} of $g_{\mathrm{BCH}_2}(x)$. Thus by Theorem 3.1 we confirm that indeed $w^*_{\mathrm{BCH}_2} \geq 7$, because it has 6 consecutive roots from F_{32}. This gives rise to search for a CRC polynomial $g_{\mathrm{CRC}}(x)$ that has β^7 as a root, because in that case the polynomial $g_{\mathrm{CRC}}(x)g_{\mathrm{BCH}_2}(x)$ will have 10 consecutive roots $(\beta, \beta^2, \ldots, \beta^{10})$ thus implying, by Theorem 3.1, that the minimum weight in the coded system is at least 11. From Table 3.3, we see that $g_{\mathrm{CRC}_{11}}(x)$ is the only CRC polynomial (of 5 bits) that has β^7 as root, therefore we immediately know that combining $g_{\mathrm{CRC}_{11}}(x)$ with $g_{\mathrm{BCH}_2}(x)$ will give a coded CRC system with minimal weight $w^* \geq 11$. If we find polynomials that have β^7 and β^{11} our code will do even better, since the minimal weight w^* in the coded system will be at least 14.

Note that using Lemma 3.1.1 we see that any polynomial that has β^7 as root, also has $\beta^{14}, \beta^{19}, \beta^{25}$ and β^{28} as root. And any polynomial that has β^{11} as roots, also has $\beta^{13}, \beta^{21}, \beta^{22}$ and β^{26} as roots. Hence we know that the CRC polynomial has to be at least of degree 10 in order to have both β^7 and β^{11} as roots. Notice that β^{14} is also the root of such polynomials, which is also consecutive to $\beta^1, \ldots, \beta^{13}$, which implies that the minimal weight of the coded system of such polynomials is at least 15. Therefore we will look at CRC polynomials of degree 10 to see whether there any of such CRC polynomials.

The CRC polynomial $g_{\mathrm{CRC}}(x) = x^{10} + x^9 + x^4 + x^3 + 1$ is the only CRC polynomial of 10 bits (which has the term 1) that has β^7 and β^{11} as roots and it generates a code that in the coded system has minimal weight $w^* = 15$. Interestingly there are two other polynomials of 10 bits that have the same weight distribution as $g_{\mathrm{CRC}}(x)$ and thus have the same undetected error probability for all values of p. One can therefore choose any of these polynomials in the coded system. To give a comparison in terms of undetected error we will look at the plot of the undetected error of $g_{\mathrm{CRC}}(x)$ compared to that of the polynomials **0x29b**, **0x28e** and **0x2b9** which are reported as good polynomials in the pure system in [8].

Notation	CRC Polynomial	Name
0x29b	$g_{\text{CRC}_1}(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$	CRC-10F/7)
0x28e	$g_{\text{CRC}_2}(x) = x^{10} + x^8 + x^4 + x^3 + x^2 + 1$	CRC-10/6sub8
0x2b9	$g_{\text{CRC}_3}(x) = x^{10} + x^8 + x^6 + x^5 + x^4 + x + 1$	CRC-10F/5
-	$g_{\text{CRC}_4}(x) = x^{10} + x^9 + x^4 + x^3 + 1$	-

Table 3.5: The polynomial representation of the CRC polynomials **0x29b**, **0x28e**, **0x2b9** and $g_{\text{CRC}_4}(x)$

As can be seen in the pure system, in Figure 3.7 depending on the value of p, the best CRC polynomial to choose is $g_{\text{CRC}_2}(x)$ for lower values of p and $g_{\text{CRC}_3}(x)$ for higher values of p. The minimum weights in the pure system are $w^*_{g_{\text{CRC}_1}} = 2$, $w^*_{g_{\text{CRC}_2}} = 6$, $w^*_{g_{\text{CRC}_3}} = 5$ and $w^*_{g_{\text{CRC}_4}} = 5$ (see Appendix A).

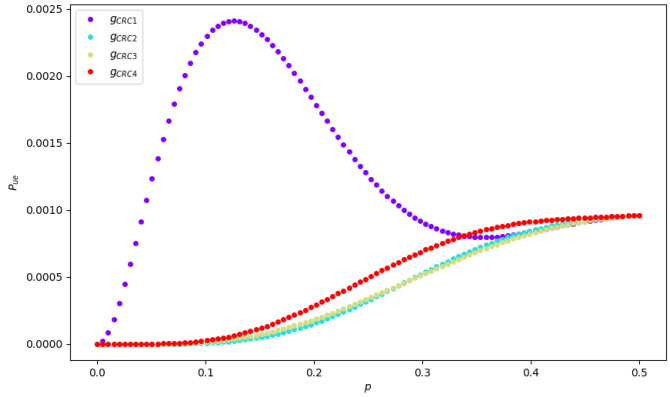

Figure 3.7: The undetected error probability P_{ue} against the channel error rate p for the $g_{\text{CRC}_1}(x), g_{\text{CRC}_2}(x), g_{\text{CRC}_3}(x)$ and $g_{\text{CRC}_4}(x)$

Instead, in the coded system, which can be seen in Figure 3.8, it is clear that $g_{\text{CRC}_4}(x)$ performs the best for all values of p. We note that the minimum weights in the coded system are $w^*_{g_{\text{CRC}_1}} = 11$, $w^*_{g_{\text{CRC}_2}} = 12$, $w^*_{g_{\text{CRC}_3}} = 11$ and $w^*_{g_{\text{CRC}_4}} = 15$ (see Appendix A).

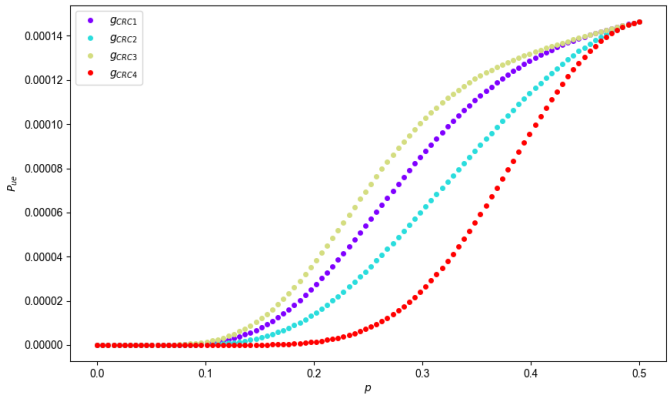

Figure 3.8: The undetected error probability P_{ue} against the channel error rate p for the $g_{\text{CRC}_1}(x), g_{\text{CRC}_2}(x), g_{\text{CRC}_3}(x)$ and $g_{\text{CRC}_4}(x)$

So also in this example we have found a CRC polynomial that performs very well in the coded system compared to other CRC polynomials that usually perform well in the pure system, but perform less good in the coded system.

Chapter 4

Conclusion and Discussion

In this chapter conclusions are given based on the results found in Chapter 3 and a discussion is given in terms of future research.

4.1 Conclusion

In this thesis the goal was to find a method in finding optimal CRC polynomials, in terms of undetected error probability in the coded system, i.e., together with BCH codes and how they relate to their performance in the pure system, based on the results that were found in [3] by K. A. Abdel-Ghaffar. In Section 3.3 it was discussed that CRC polynomials of degree 5 that tend to perform well in the pure system might not perform well in the coded system, i.e. in combination with a BCH code, and vice versa. In Section 3.4 a method was created to find good CRC polynomials in combination with BCH codes by looking at the amount of consecutive roots of these polynomials and comparing them to those of the BCH code. CRC polynomials that have many roots that are consecutive to those of the BCH generator polynomial tend to perform better since they generate codes that have higher minimum weight. For the coded system, one can select the best CRC polynomial by looking at all the consecutive roots of the product of the BCH generator polynomial and the CRC polynomial, since it will give a lower bound on the minimum weight. One does need to be careful since in some cases the actual minimal weight of a coded CRC system, might be higher than the lower bound that Theorem 3.1 gives, as was seen in Section 3.4 where $g_{\mathrm{CRC}_{16}}(x)$ had a lower bound of 5 on the minimal weight, whereas the actual minimal weight is $w^* = 7$, therefore besides looking at the amount of consecutive roots it is strongly advised to generate the weight distributions of the different CRC polynomials in the coded system, since the lower bound on the weight is not always a tight bound.

4.2 Discussion

Based on the conclusion given in the previous section, the actual minimal weight of a code is not always equal to the lower bound that was given by Theorem 3.1, therefore further research into the weight distributions is recommended, one might focus on trying to find good upper bounds to limit the search of CRC polynomials and focus on the ones that have tight and high bounds, or find special conditions in which CRC polynomials have minimal weight equal to the lower bound provided by Theorem 3.1. As noted in 3.4, CRC polynomials which have the best minimal weight in the coded system might also perform less optimal when increasing the value of p, future research is advised to look into the influence of specific weights, or weight amounts on performance of a CRC polynomial in terms of undetected error probability in combination with the channel error rate

p. Some CRC polynomials have an undetected error probability function that decreases when the channel error rate is high enough, further research needs to be done to give an explanation for when the function behaves in this way, because it might also be useful in finding a good CRC polynomil. This thesis mainly focused on BCH codes, given their handy properties such as having a generator consisting of products of minimal polynomials, which immediately gives us control on the amount of consecutive roots and consequently the minimal distance of the code. However it is advised that future research be done on the CRC systems in combination with other error correcting codes such as the Golay code or Reed-Muller codes [14], given that the minimal distance of these codes is known and since these codes have other handy properties that might be of use when combining with CRC polynomials.

In this thesis a method was given to try and find optimal CRC polynomials for CRC systems in combination with error-correcting codes, some improvements have definitely been made based on this method since it selects codes that have higher lower bounds on the minimal weight, however more research needs to be done to improve these methods.

References

[1] M. V. Patil, S. Pawar, and Z. Saquib, (2020), Coding techniques for 5G networks: A Review. *2020 3rd International Conference on Communication System, Computing and IT Applications (CSCITA).*

[2] Goodin, D. (2021, March 4). Bitflips when PCs try to reach windows.com: What could possibly go wrong? *Ars Technica.* https://arstechnica.com/gadgets/2021/03/windows-com-bitsquatting-hack-can-wreak-unknown-havoc-on-pcs/

[3] K. A. Abdel-Ghaffar, "CRC in coded schemes with bounded-distance decoding (2023)," *2023 IEEE Wireless Communications and Networking Conference (WCNC).*

[4] Bose, R.C., Ray-Chaudhuri, D.K. (March 1960), "On A Class of Error Correcting Binary Group Codes", *Information and Control*

[5] Hughes, T. (2020, August 10). *The Vandermonde Determinant, A Novel Proof.* Towards Data Science. https://towardsdatascience.com/the-vandermonde-determinant-a-novel-proof-851d107bd728

[6] Veloce. (2023). What Is a Burst Error? *Veloce Network.* https://www.velocenetwork.com/tech/what-is-a-burst-error/#Impact_of_Burst_Errors_on_Data_Transmission_and_Storage

[7] Hickerson, A. (n.d.). *The POCSAG paging protocol.* Raveon. https://www.raveon.com/pdfiles/AN142(POCSAG).pdf

[8] Koopman, P. (2018). *Best CRC Polynomials.* https://users.ece.cmu.edu/~koopman/crc/

[9] Koopman, P. (2010) What's the best CRC polynomial to use? *Better Embedded System SW* https://betterembsw.blogspot.com/

[10] Baicheva, T. (2008) Determination of the Best CRC Codes with up to 10-Bit Redundancy. *IEEE Transactions on Communications*

[11] Cyclic redundancy checks in USB (n.d.) *USB* Retrieved May 31, 2023 from https://www.usb.org/sites/default/files/crcdes.pdf

[12] Soukthavy, S. (n.d.) *Binary BCH (31,16,7) linear cyclic code work out.* https://souktha.github.io/misc/bch31_16_7/

[13] Reed, I.S., Chen, X. (1999). BCH Codes. In: Error-Control Coding for Data Networks. *The Springer International Series in Engineering and Computer Science*, vol 508. 189-231 Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-5005-1_5

[14] D. E. Muller, (September 1954) Application of Boolean algebra to switching circuit design and to error detection, *Transactions of the I.R.E. Professional Group on Electronic Computers*, vol. EC-3, no. 3, pp. 6-12, Sept. 1954, doi: 10.1109/IREPGELC.1954.6499441.

Appendix A

Weight Distributions

A.1 [31,21,5] BCH code

In this Section we will look at the weight distributions for the CRC polynomials of degree 5 in the pure system and together with the [31,21,5] BCH code. In Section A.1.1 the weight distributions are given for the CRC polynomials in the pure system, i.e., the CRC polynomials of degree 5 that append to 16 bits of data. In Section A.1.2 the weight distributions are given when combining these CRC polynomials with the [31,21,5] BCH code.

A.1.1 Pure system

w	$A_w^{g_{\mathrm{CRC}_1}}$	$A_w^{g_{\mathrm{CRC}_2}}$	$A_w^{g_{\mathrm{CRC}_3}}$	$A_w^{g_{\mathrm{CRC}_4}}$	$A_w^{g_{\mathrm{CRC}_5}}$	$A_w^{g_{\mathrm{CRC}_6}}$	$A_w^{g_{\mathrm{CRC}_7}}$	$A_w^{g_{\mathrm{CRC}_8}}$
1	0	0	0	0	0	0	0	0
2	6	0	0	0	7	9	18	27
3	0	45	47	42	0	0	0	0
4	397	205	205	210	388	384	408	300
5	0	632	616	651	0	0	0	0
6	3352	1672	1672	1638	3388	3360	3196	3492
7	0	3620	3676	3570	0	0	0	0
8	12754	6370	6370	6468	12670	12810	12978	12774
9	0	9240	9128	9310	0	0	0	0
10	22036	11032	11032	10878	22162	21882	22032	21762
11	0	10934	11074	10878	0	0	0	0
12	18354	9170	9170	9310	18228	18536	18192	18668
13	0	6440	6328	6468	0	0	0	0
14	7288	3640	3640	3570	7372	7176	7332	7140
15	0	1652	1708	1638	0	0	0	0
16	1261	637	637	651	1225	1293	1317	1281
17	0	200	184	210	0	0	0	0
18	86	40	40	42	95	85	62	91
19	0	5	7	0	0	0	0	0
20	1	1	1	0	0	0	0	0
21	0	0	0	1	0	0	0	0

Table A.1: The weight distributions of the pure CRC system for the first eight CRC polynomials

w	$A_w^{g_{\mathrm{CRC}_9}}$	$A_w^{g_{\mathrm{CRC}_{10}}}$	$A_w^{g_{\mathrm{CRC}_{11}}}$	$A_w^{g_{\mathrm{CRC}_{12}}}$	$A_w^{g_{\mathrm{CRC}_{13}}}$	$A_w^{g_{\mathrm{CRC}_{14}}}$	$A_w^{g_{\mathrm{CRC}_{15}}}$	$A_w^{g_{\mathrm{CRC}_{16}}}$
1	0	0	0	0	0	0	0	0
2	7	6	0	0	0	0	34	0
3	0	0	47	47	45	42	0	47
4	388	397	205	204	205	210	465	204
5	0	0	616	616	632	651	0	616
6	3388	3352	1672	1680	1672	1638	3256	1680
7	0	0	3676	3676	3620	3570	0	3676
8	12670	12754	6370	6342	6370	6468	12194	6342
9	0	0	9128	9128	9240	9310	0	9128
10	22162	22036	11032	11088	11032	10878	22956	11088
11	0	0	11074	11074	10934	10878	0	11074
12	18228	18354	9170	9100	9170	9310	18250	9100
13	0	0	6328	6328	6440	6468	0	6328
14	7372	7288	3640	3696	3640	3570	6904	3696
15	0	0	1708	1708	1652	1638	0	1708
16	1225	1261	637	609	637	651	1341	609
17	0	0	184	184	200	210	0	184
18	95	86	40	48	40	42	130	48
19	0	0	7	7	5	0	0	7
20	0	1	1	0	1	0	5	0
21	0	0	0	0	0	1	0	0

Table A.2: The weight distributions of the pure CRC system for the last eight CRC polynomials

A.1.2 Coded scheme

w	$A_w^{g_{\mathrm{CRC}_1}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_2}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_3}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_4}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_5}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_6}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_7}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_8}}(\mathcal{C})$
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	6	6	0	0	0	0	0
6	29	18	27	23	53	57	66	46
7	0	85	67	95	0	0	0	0
8	543	233	199	233	477	450	474	478
9	0	591	655	504	0	0	0	0
10	2553	1340	1387	1325	2599	2646	2501	2621
11	0	2640	2592	2628	0	0	0	0
12	8944	4496	4464	4452	8936	8944	9048	8968
13	0	6164	6084	6100	0	0	0	0
14	15706	7724	7734	7834	15714	15626	15820	15636
15	0	9426	9566	9498	0	0	0	0
16	18791	9467	9495	9471	18859	18917	18653	18805
17	0	7790	7758	7830	0	0	0	0
18	12282	6192	6126	6062	12158	12180	12186	12282
19	0	4504	4440	4420	0	0	0	0
20	5344	2608	2656	2668	5368	5336	5456	5376
21	0	1294	1342	1316	0	0	0	0
22	1161	594	591	623	1209	1221	1162	1142
23	0	241	231	255	0	0	0	0
24	169	91	81	71	151	144	160	164
25	0	27	21	21	0	0	0	0
26	13	7	5	5	11	14	9	17
27	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0

Table A.3: The weight distributions of the coded CRC system for the first eight CRC polynomials

w	$A_w^{g_{\mathrm{CRC}_9}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_{10}}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_{11}}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_{12}}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_{13}}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_{14}}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_{15}}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_{16}}}(\mathcal{C})$
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	31	9	0	0
6	49	39	31	0	0	11	42	0
7	0	0	0	155	0	72	0	155
8	477	492	310	465	310	247	507	465
9	0	0	775	0	620	597	0	0
10	2619	2658	1116	0	1271	1365	2552	0
11	0	0	2852	5208	2852	2684	0	5208
12	8936	8864	4340	8680	4340	4412	9040	8680
13	0	0	5580	0	5890	6094	0	0
14	15674	15638	8370	0	8060	7770	15596	0
15	0	0	9393	18259	9393	9436	0	18259
16	18859	18969	9393	18259	9393	9523	18815	18259
17	0	0	8370	0	8060	7838	0	0
18	12198	12188	5580	0	5890	6118	12328	0
19	0	0	4340	8680	4340	4468	0	0
20	5368	5304	2852	5208	2852	2628	5280	5208
21	0	0	1116	0	1271	1305	0	0
22	1189	1211	775	0	620	603	1210	0
23	0	0	310	465	310	236	0	465
24	151	162	0	155	0	85	149	155
25	0	0	31	0	0	29	0	0
26	15	10	0	0	31	5	16	0
27	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0
31	0	0	1	0	1	0	0	1

Table A.4: The weight distributions of the coded CRC system for the last eight CRC polynomials

A.2 [31,16,7] BCH code

In this section the weight distributions are given of the polynomials , **0x29b**, **0x28e** and **0x2b9**, $g_{\mathrm{CRC}_4}(x)$ denoted respectively by $g_{\mathrm{CRC}_1}(x)$, $g_{\mathrm{CRC}_2}(x)$, $g_{\mathrm{CRC}_3}(x)$ and $g_{\mathrm{CRC}_4}(x)$, in the pure system and the coded system with the [31,16,7] BCH code generated by $g_{\mathrm{BCH}_2}(x)$ as discussed in Section 3.5. In Section A.2.1 the weight distributions are given for these CRC polynomials in the pure system. In Section A.2.2 the weight distributions are given for these CRC polynomials in combination with the [31,16,7] code.

A.2.1 Pure scheme

w	$A_w^{g_{\mathrm{CRC}_1}}$	$A_w^{g_{\mathrm{CRC}_2}}$	$A_w^{g_{\mathrm{CRC}_3}}$	$A_w^{g_{\mathrm{CRC}}}$
2	1	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	4	7
6	0	21	6	10
7	22	0	13	11
8	23	22	17	9
9	8	0	11	7
10	7	15	6	10
11	0	0	3	5
12	0	5	2	2
13	0	0	1	2
14	0	0	0	0
15	2	0	0	0

Table A.5: The weight distributions of the pure CRC system for $g_{\mathrm{CRC}_1}(x)$,$g_{\mathrm{CRC}_2}(x)$, $g_{\mathrm{CRC}_3}(x)$ and $g_{\mathrm{CRC}_4}(x)$

A.2.2 Coded scheme

w	$A_w^{g_{\mathrm{CRC}_1}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_2}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_3}}(\mathcal{C})$	$A_w^{g_{\mathrm{CRC}_4}}(\mathcal{C})$
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	4	0	8	0
12	8	11	4	0
13	0	0	0	0
14	0	0	0	0
15	20	0	12	31
16	16	45	23	31
17	0	0	0	0
18	0	0	0	0
19	8	0	12	0
20	6	7	4	0
21	0	0	0	0
22	0	0	0	0
23	0	0	0	0
24	1	0	0	0
25	0	0	0	0
26	0	0	0	0
27	0	0	0	0
28	0	0	0	0
29	0	0	0	0
30	0	0	0	0
31	0	0	0	1

Table A.6: The weight distributions of the coded CRC system for $g_{\mathrm{CRC}_1}(x)$,$g_{\mathrm{CRC}_2}(x)$, $g_{\mathrm{CRC}_3}(x)$ and $g_{\mathrm{CRC}_4}(x)$

Appendix B

Mapping Analysis

In this part of the appendix the results are shown for the mapping analysis described in Section 3.4. In the first column the weight of the CRC words are given. In the second column the weights that the code words have in the coded system obtained by mapping the CRC words of fixed weight to the BCH code. In each table the minimum weight in the coded system is highlighted in red

B.1 $g_{\mathbf{CRC}_1}(x) = x^5 + x^3 + x + 1$ **and**
$g_{\mathbf{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$

CRC code	**BCH code**	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\text{CRC}} = 2$	$w = 14$	6
$w_{\text{CRC}} = 4$	$w = 8$	42
	$w = 10$	61
	$w = 12$	77
	$w = 14$	88
	$w = 16$	85
	$w = 18$	34
	$w = 20$	10
$w_{\text{CRC}} = 6$	$w = 8$	106
	$w = 10$	222
	$w = 12$	716
	$w = 14$	908
	$w = 16$	771
	$w = 18$	450
	$w = 20$	146
	$w = 22$	31
	$w = 24$	2
$w_{\text{CRC}} = 8$	$w^* = 6$	9
	$w = 8$	153
	$w = 10$	709
	$w = 12$	2170
	$w = 14$	3267
	$w = 16$	3413
	$w = 18$	2026
	$w = 20$	815
	$w = 22$	168
	$w = 24$	24
$w_{\text{CRC}} = 10$	$w^* = 6$	16
	$w = 8$	149
	$w = 10$	851
	$w = 12$	2946
	$w = 14$	5425
	$w = 16$	6402
	$w = 18$	4104
	$w = 20$	1752
	$w = 22$	346
	$w = 24$	40
	$w = 26$	5

Table B.1: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\text{CRC}_1}(x) = x^5 + x^3 + x + 1$ together with $g_{\text{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part A)

CRC code	**BCH code**	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 12$	$w^* = 6$	4
	$w = 8$	84
	$w = 10$	513
	$w = 12$	2199
	$w = 14$	4192
	$w = 16$	5479
	$w = 18$	3711
	$w = 20$	1700
	$w = 22$	394
	$w = 24$	71
	$w = 26$	7
$w_{\mathrm{CRC}} = 14$	$w = 8$	7
	$w = 10$	165
	$w = 12$	708
	$w = 14$	1564
	$w = 16$	2254
	$w = 18$	1629
	$w = 20$	752
	$w = 22$	179
	$w = 24$	30
$w_{\mathrm{CRC}} = 16$	$w = 8$	2
	$w = 10$	31
	$w = 12$	120
	$w = 14$	245
	$w = 16$	355
	$w = 18$	307
	$w = 20$	157
	$w = 22$	41
	$w = 24$	2
	$w = 26$	1
$w_{\mathrm{CRC}} = 18$	$w = 10$	1
	$w = 12$	8
	$w = 14$	11
	$w = 16$	31
	$w = 18$	21
	$w = 20$	12
	$w = 22$	2
$w_{\mathrm{CRC}} = 20$	$w = 16$	1

Table B.2: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_1}(x) = x^5 + x^3 + x + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part B)

B.2 $g_{\mathbf{CRC}_2}(x) = x^5 + x^2 + 1$ and $g_{\mathbf{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$

CRC code	BCH code	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 3$	$w = 13$	16
	$w = 15$	21
	$w = 17$	5
	$w = 19$	3
$w_{\mathrm{CRC}} = 4$	$w = 8$	27
	$w = 10$	47
	$w = 12$	13
	$w = 14$	58
	$w = 16$	50
	$w = 18$	10
$w_{\mathrm{CRC}} = 5$	$w = 9$	65
	$w = 11$	101
	$w = 13$	164
	$w = 15$	191
	$w = 17$	70
	$w = 19$	33
	$w = 21$	8
$w_{\mathrm{CRC}} = 6$	$w = 8$	29
	$w = 10$	148
	$w = 12$	403
	$w = 14$	322
	$w = 16$	387
	$w = 18$	250
	$w = 20$	102
	$w = 22$	19
	$w = 24$	6
$w_{\mathrm{CRC}} = 7$	$w = 7$	21
	$w = 9$	153
	$w = 11$	420
	$w = 13$	750
	$w = 15$	1102
	$w = 17$	756
	$w = 19$	310
	$w = 21$	93
	$w = 23$	14
	$w = 25$	1
$w_{\mathrm{CRC}} = 8$	$w^* = 6$	3
	$w = 8$	43
	$w = 10$	415
	$w = 12$	1041
	$w = 14$	1635
	$w = 16$	1691
	$w = 18$	1079
	$w = 20$	401
	$w = 22$	51
	$w = 24$	11

Table B.3: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_2}(x) = x^5 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part A)

CRC code	BCH code	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 9$	$w = 7$	41
	$w = 9$	171
	$w = 11$	877
	$w = 13$	1896
	$w = 15$	2649
	$w = 17$	2091
	$w = 19$	1143
	$w = 21$	314
	$w = 23$	55
	$w = 25$	3
$w_{\mathrm{CRC}} = 10$	$w = 6$	8
	$w = 8$	90
	$w = 10$	370
	$w = 12$	1542
	$w = 14$	2710
	$w = 16$	3230
	$w = 18$	2014
	$w = 20$	846
	$w = 22$	198
	$w = 24$	23
	$w = 26$	1
$w_{\mathrm{CRC}} = 11$	$w^* = 5$	6
	$w = 7$	13
	$w = 9$	135
	$w = 11$	779
	$w = 13$	1989
	$w = 15$	3115
	$w = 17$	2691
	$w = 19$	1639
	$w = 21$	464
	$w = 23$	89
	$w = 25$	14
$w_{\mathrm{CRC}} = 12$	$w = 8$	32
	$w = 10$	261
	$w = 12$	1109
	$w = 14$	2076
	$w = 16$	2798
	$w = 18$	1863
	$w = 20$	790
	$w = 22$	206
	$w = 24$	32
	$w = 26$	3

Table B.4: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_2}(x) = x^5 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part C)

CRC code	BCH code	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 13$	$w = 7$	10
	$w = 9$	53
	$w = 11$	360
	$w = 13$	1076
	$w = 15$	1869
	$w = 17$	1677
	$w = 19$	1016
	$w = 21$	310
	$w = 23$	64
	$w = 25$	5
$w_{\mathrm{CRC}} = 14$	$w = 6$	1
	$w = 8$	12
	$w = 10$	95
	$w = 12$	319
	$w = 14$	794
	$w = 16$	1104
	$w = 18$	807
	$w = 20$	384
	$w = 22$	106
	$w = 24$	18
$w_{\mathrm{CRC}} = 15$	$w = 9$	13
	$w = 11$	91
	$w = 13$	254
	$w = 15$	432
	$w = 17$	435
	$w = 19$	317
	$w = 21$	89
	$w = 23$	17
	$w = 25$	4
$w_{\mathrm{CRC}} = 16$	$w = 10$	4
	$w = 12$	61
	$w = 14$	117
	$w = 16$	201
	$w = 18$	158
	$w = 20$	81
	$w = 22$	14
	$w = 24$	1
$w_{\mathrm{CRC}} = 17$	$w = 9$	1
	$w = 11$	12
	$w = 13$	18
	$w = 15$	47
	$w = 17$	64
	$w = 19$	42
	$w = 21$	14
	$w = 23$	2

Table B.5: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_2}(x) = x^5 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part D)

CRC code	**BCH code**	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 18$	$w = 12$	8
	$w = 14$	12
	$w = 16$	5
	$w = 18$	11
	$w = 20$	4
$w_{\mathrm{CRC}} = 19$	$w = 13$	1
	$w = 17$	1
	$w = 19$	1
	$w = 21$	2
$w_{\mathrm{CRC}} = 20$	$w = 16$	1

Table B.6: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_2}(x) = x^5 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part E)

B.3 $g_{\mathbf{CRC}_3}(x) = x^5 + x^4 + x^3 + x^2 + 1$ and $g_{\mathbf{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$

CRC code	**BCH code**	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 3$	$w = 9$	13
	$w = 11$	12
	$w = 15$	15
	$w = 17$	7
$w_{\mathrm{CRC}} = 4$	$w = 8$	14
	$w = 10$	30
	$w = 12$	33
	$w = 14$	22
	$w = 16$	87
	$w = 18$	17
	$w = 20$	2
$w_{\mathrm{CRC}} = 5$	$w = 9$	60
	$w = 11$	90
	$w = 13$	179
	$w = 15$	139
	$w = 17$	93
	$w = 19$	45
	$w = 21$	9
	$w = 23$	1
$w_{\mathrm{CRC}} = 6$	$w = 6$	12
	$w = 8$	46
	$w = 10$	154
	$w = 12$	350
	$w = 14$	425
	$w = 16$	418
	$w = 18$	183
	$w = 20$	77
	$w = 22$	7
$w_{\mathrm{CRC}} = 7$	$w = 7$	21
	$w = 9$	103
	$w = 11$	468
	$w = 13$	908
	$w = 15$	969
	$w = 17$	715
	$w = 19$	370
	$w = 21$	110
	$w = 23$	11
	$w = 25$	1

Table B.7: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_3}(x) = x^5 + x^4 + x^3 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part A)

CRC code	BCH code	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 8$	$w = 6$	6
	$w = 8$	52
	$w = 10$	412
	$w = 12$	1020
	$w = 14$	1475
	$w = 16$	1861
	$w = 18$	1019
	$w = 20$	434
	$w = 22$	77
	$w = 24$	14
$w_{\mathrm{CRC}} = 9$	$w = 7$	33
	$w = 9$	240
	$w = 11$	741
	$w = 13$	1842
	$w = 15$	2742
	$w = 17$	2070
	$w = 19$	1080
	$w = 21$	320
	$w = 23$	55
	$w = 25$	5
$w_{\mathrm{CRC}} = 10$	$w = 6$	8
	$w = 8$	41
	$w = 10$	473
	$w = 12$	1537
	$w = 14$	2704
	$w = 16$	3221
	$w = 18$	2017
	$w = 20$	816
	$w = 22$	192
	$w = 24$	20
	$w = 26$	3
$w_{\mathrm{CRC}} = 11$	$w^* = 5$	6
	$w = 7$	9
	$w = 9$	154
	$w = 11$	793
	$w = 13$	1945
	$w = 15$	3328
	$w = 17$	2693
	$w = 19$	1575
	$w = 21$	463
	$w = 23$	92
	$w = 25$	16

Table B.8: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_3}(x) = x^5 + x^4 + x^3 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part B)

CRC code	**BCH code**	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 12$	$w = 8$	30
	$w = 10$	222
	$w = 12$	1084
	$w = 14$	2184
	$w = 16$	2657
	$w = 18$	1900
	$w = 20$	862
	$w = 22$	196
	$w = 24$	34
	$w = 26$	1
$w_{\mathrm{CRC}} = 13$	$w = 7$	4
	$w = 9$	78
	$w = 11$	388
	$w = 13$	969
	$w = 15$	1823
	$w = 17$	1657
	$w = 19$	1029
	$w = 21$	323
	$w = 23$	53
	$w = 25$	4
$w_{\mathrm{CRC}} = 14$	$w = 6$	1
	$w = 8$	14
	$w = 10$	83
	$w = 12$	383
	$w = 14$	803
	$w = 16$	1028
	$w = 18$	828
	$w = 20$	396
	$w = 22$	90
	$w = 24$	12
	$w = 26$	2
$w_{\mathrm{CRC}} = 15$	$w = 9$	7
	$w = 11$	88
	$w = 13$	216
	$w = 15$	501
	$w = 17$	469
	$w = 19$	302
	$w = 21$	106
	$w = 23$	18
	$w = 25$	1
$w_{\mathrm{CRC}} = 16$	$w = 8$	2
	$w = 10$	11
	$w = 12$	54
	$w = 14$	120
	$w = 16$	203
	$w = 18$	151
	$w = 20$	66
	$w = 22$	28
	$w = 24$	1
	$w = 26$	1

Table B.9: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_3}(x) = x^5 + x^4 + x^3 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part C)

CRC code	**BCH code**	
CRC words of weight	**Weights BCH code**	**Amount**
$w_{\mathrm{CRC}} = 17$	$w = 11$	12
	$w = 13$	23
	$w = 15$	49
	$w = 17$	53
	$w = 19$	35
	$w = 21$	11
	$w = 23$	1
$w_{\mathrm{CRC}} = 18$	$w = 10$	2
	$w = 12$	3
	$w = 14$	1
	$w = 16$	20
	$w = 18$	11
	$w = 20$	2
	$w = 22$	1
$w_{\mathrm{CRC}} = 19$	$w = 13$	3
	$w = 17$	1
	$w = 19$	4
$w_{\mathrm{CRC}} = 20$	$w = 20$	1

Table B.10: The weight distribution of the coded scheme generated by the CRC words of a fixed weight, using $g_{\mathrm{CRC}_3}(x) = x^5 + x^4 + x^3 + x^2 + 1$ together with $g_{\mathrm{BCH}}(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ (part D)

Appendix C

Field Table

0	00000	β^7	10100	β^{15}	11111	β^{23}	01111
1	00001	β^8	01101	β^{16}	11011	β^{24}	11110
β	00010	β^9	11010	β^{17}	10011	β^{25}	11001
β^2	00100	β^{10}	10001	β^{18}	00011	β^{26}	10111
β^3	01000	β^{11}	00111	β^{19}	00110	β^{27}	01011
β^4	10000	β^{12}	01110	β^{20}	01100	β^{28}	10110
β^5	00101	β^{13}	11100	β^{21}	11000	β^{29}	01001
β^6	01010	β^{14}	11101	β^{22}	10101	β^{30}	10010

Table C.1: F_{32} with primitive element β such that $\beta^5 = \beta^2 + 1$

Appendix D

Python Code

D.1 weights_finder_pure.py

```
from numpy.polynomial import polynomial as P
import numpy as np
import math
import itertools

def add_bin(a,b):
     return (a + b) % 2

def multiply(p, q, n):
     res = P.polymul(p,q)
     res %= 2
     return np.pad(res, (0,n-len(res)))


## the following finds the weights distribution of a code using
## the generator polynomial
## generator polynomial has degree r = n-k
## so every product of the generator polynomial with a polynomial
## of degree <= n-r is a code-word

def code_gen(gen, k):
     code = []
     r = len(gen)
     lst = [list(i) for i in itertools.product([0,1], repeat = k-r+1)]
     del lst[0]
     for pol in lst:
          word = multiply(gen, pol,k)
          code.append(word)
     return code


## create a subset of the code of only the words that have the specified weight
def find_subset(code, weight):
     subset = []
```

```
     for word in code:
          if sum(word) == weight:
               subset.append(word)
     return subset

## Find the weights of any pre-generated code

def w_finder(code):
     weights = {}
     res = 0
     for word in code:
          w = int(sum(word))
          if w in weights:
               weights[w] += 1
          else:
               weights[w] = 1
     return weights

def find_min_weight(weights):
     res = float('inf')
     for weight in weights:
          if weight < res:
               res = weight
     return res
```

D.2 weights_finder_coded_.py

```
import math
import numpy as np
from weights_finder_pure import *

## given a generator polynomial return generator matrix ##

def gen_matrix(gen, n):
    r = len(gen)-1
    k = n-r
    G = np.zeros((k,n))
    for i in range(k):
        for j in range(r+1):
            G[i][i+j] = int(gen[j]) % 2
    return G

## find the weights of the coded scheme using a generator polynomial

def w_finder_coded(gen,n,g_crc,k):
    G = gen_matrix(gen,n)
    code = code_gen(g_crc,k)
    code_new = [code[i].dot(G)%2 for i in range(len(code))]
    weights = w_finder(code_new)
    return weights
```

```
## find the weights of the coded scheme using the already generated CRC code

def w_finder_coded_2(gen,n,code):
    G = gen_matrix(gen,n)
    code_new = [code[i].dot(G)%2 for i in range(len(code))]
    weights = w_finder(code_new)
    return weights

## Find the weight distribution of the code
## that is generated by a subset of a CRC code of a certain weight

def partition(gen,n,crc,k):
    w_lst = []
    for weight in range(k+1):
        code = find_subset(crc,weight)
        weights = w_finder_coded_2(gen,n,code)
        w_lst.append(weights)
    return w_lst
```

D.3 undetected_error.py

```
import matplotlib.pyplot as plt
import numpy as np
from math import comb
from weights_finder_pure import *
from weights_finder_coded import *
from matplotlib.pyplot import cm

## Calculate the undetected error for the pure system

def pue_pure(pch, weights,k):
    p_ue = 0
    for w in weights:
        value = weights[w]
        p_ue += value * (pch ** w) * (1-pch) ** (k-w)
    return p_ue

## Calculate the undetected error for the coded system

def pue_coded(t, pch, weights, n):
    p_ue = 0
    for w in weights:
        value = weights[w]
        t_0 = 0
        t_1 = 0
        while t_0 + t_1 <= t:
            part_sum = value * comb(w,t_0) * comb(n-w,t_1) * \
```

```
                    pch**(w + t_1 - t_0) * (1-pch)**(n-w-t_1+t_0)
            p_ue += part_sum
            if t_0 + t_1 == t:
                t_1 += 1
                t_0 = 0
            else:
                t_0 += 1
    return p_ue

## Generate a single plot of the undetected error of a CRC polynomial
## In the pure system for all values of p

def single_plot_pure(weights, k):
    fig = plt.figure()
    xpoints =  np.linspace(0, 0.5,51)
    y = []
    for pch in xpoints:
        a = pue_pure(pch, weights, k)
        y.append(a)
    plt.plot(xpoints, y, 'o')
    plt.show()

## Generate a plot of the undetected error of a list of CRC polynomials
## In the pure system for all values of p

def full_plot_pure(w_lst, k):
    fig = plt.figure(figsize=(10,6))
    xpoints =  np.linspace(0, 0.5, 100)
    y = []
    i = 1
    color = iter(cm.rainbow(np.linspace(0,1,len(w_lst))))
    for weights in w_lst:
        y_1 = []
        for pch in xpoints:
            a = pue_pure(pch, weights, k)
            y_1.append(a)
        y.append(y_1)
    for el in y:
        c = next(color)
        plt.plot(xpoints, el, 'o',label = '$g_{CRC%s}$' % i, markersize = 4, c = c)
        i+=1
    plt.xlabel('$p$')
    plt.ylabel('$P_{ue}$')
    plt.legend(loc = "upper left")
    plt.savefig('pure_plot.png')
    plt.show()

## Generate a plot of the undetected error of a CRC polynomial
## In the coded system for all values of p
```

```
def single_plot(t, weights, n):
    fig = plt.figure()
    xpoints =  np.linspace(0, 0.5,51)
    y = []
    for pch in xpoints:
        a = pue_coded(t, pch, weights, n)
        y.append(a)
    plt.plot(xpoints, y, 'o')
    plt.show()

## Generate a plot of the undetected error of a list of CRC polynomials
## In the coded system for all values of p

def full_plot(t, w_lst, n):
    fig = plt.figure(figsize=(10,6))
    xpoints =  np.linspace(0, 0.5, 100)
    y = []
    i = 1
    color = iter(cm.rainbow(np.linspace(0,1,len(w_lst))))
    for weights in w_lst:
        y_1 = []
        for pch in xpoints:
            a = pue_coded(t, pch, weights, n)
            y_1.append(a)
        y.append(y_1)
    for el in y:
        c = next(color)
        plt.plot(xpoints, el, 'o', label = '$g_{CRC%s}$' % i,markersize = 4, c = c)
        i+=1
    plt.xlabel('$p$')
    plt.ylabel('$P_{ue}$')
    plt.legend(loc = "upper left")
    plt.style.use('seaborn-v0_8')
    plt.savefig('coded_plot.png')
    plt.show()
```

D.4 calculate_roots.py

```
import numpy as np
import galois
import itertools
from weights_finder_pure import add_bin, multiply
from numpy.polynomial import polynomial as P

## The field F_{32} as binary arrays of 5 bits
gf = np.array([[0,0,0,0,1],\
      [0,0,0,1,0],\
      [0,0,1,0,0],\
      [0,1,0,0,0],\
      [1,0,0,0,0],\
```

```
       [0,0,1,0,1],\
       [0,1,0,1,0],\
       [1,0,1,0,0],\
       [0,1,1,0,1],\
       [1,1,0,1,0],\
       [1,0,0,0,1],\
       [0,0,1,1,1],\
       [0,1,1,1,0],\
       [1,1,1,0,0],\
       [1,1,1,0,1],\
       [1,1,1,1,1],\
       [1,1,0,1,1],\
       [1,0,0,1,1],\
       [0,0,0,1,1],\
       [0,0,1,1,0],\
       [0,1,1,0,0],\
       [1,1,0,0,0],\
       [1,0,1,0,1],\
       [0,1,1,1,1],\
       [1,1,1,1,0],\
       [1,1,0,0,1],\
       [1,0,1,1,1],\
       [0,1,0,1,1],\
       [1,0,1,1,0],\
       [0,1,0,0,1],\
       [1,0,0,1,0]])


## Create a list of all possible CRC polynomials of k bits, that have +1
## As a term

def create_crc_list(k):
    res = []
    lst = [list(i) for i in itertools.product([0,1], repeat = k-1)]
    for i in range(len(lst)):
        res.append([1] + lst[i] + [1])
    return np.array(res)

## find g(beta**power) for any power
def check_root(g,power):
        res = np.array([0]*5)
        if power == 0:
            res[-1] = sum(g)%2
            return res
        else:
            for i in range(len(g)):
                new_pow = (power * i)%31
                res += gf[new_pow]*g[i]
        return res%2
```

```
## find all the roots of a generator polynomial
def return_roots(g):
    root_powers = []
    for i in range(len(gf)):
        if sum(check_root(g, i)) == 0:
            root_powers.append(i)
    return root_powers

###################################

def main():
    ## Find the roots of the CRC polynomials of 5 bits
    crc_5 = create_crc_list(5)
    for crc in crc_5:
        print(np.poly1d(crc[::-1]))
        beta_lst = return_roots(crc)
        print(beta_lst)
        print('\n')

    ## Find the the CRC polynomials of 10 bits that have beta^7 and
    ## beta^11 as roots

    for i in range(1,16):
        print('\n', "CRC of %s bits:"%i, '\n')
        crc_i = create_crc_list(i)
        for crc in crc_i:
            beta_lst = return_roots(crc)
            if len(beta_lst) >= 10 and (7 in beta_lst) and (11 in beta_lst):
                print(np.poly1d(crc[::-1]))
                print(beta_lst)
                print('\n')


if __name__ == "__main__":
    main()
```

D.5 subset_crc_code.py

```
from weights_finder_pure import *
from weights_finder_coded import *
import numpy as np
import math
import itertools

## My own example with 5 CRC bits
g1 = [1,1,0,1,0,1] #0x15 HD = 2 ----> w* = 6
g2 = [1,0,1,0,0,1] #0x12 HD = 3 ----> w* = 5
g3 = [1,0,1,1,1,1] #0x1e HD = 3 ----> w* = 5
##glst = [g1,g2,g3]
n,k,t = 31,21,2
```

```
g_C = [1,0,0,1,0,1,1,0,1,1,1]
##g_C = [1,1,1,1,0,1,0,1,1,1,1,1,1,0,0,0,1]

crc1 = code_gen(g1,k)
crc2 = code_gen(g2,k)
crc3 = code_gen(g3,k)

subset_lst = partition(g_C,n,crc1,k)
subset_lst2 = partition(g_C,n,crc2,k)
subset_lst3 = partition(g_C,n,crc3,k)
print(subset_lst,'\n')
print(subset_lst2,'\n')
print(subset_lst3,'\n')
```

D.6 main.py

```
from undetected_error import *
from calculate_roots import create_crc_list
from weights_finder_coded import *
from weights_finder_pure import *
from numpy.polynomial import polynomial as P

#### BCH code 1 #####
n,k,t = 31,21,2
g_C = [1,0,0,1,0,1,1,0,1,1,1]

g1 = [1,1,0,1,0,1]
g2 = [1,0,1,0,0,1]
g3 = [1,0,1,1,1,1]
g12 = [1,1,1,0,1,1]

### First example with 3 CRC polynomials
glst = [g1,g2,g3]

## Second example with 4 g_CRC_12(x) compared to the other 3
glst2 = [g1,g2,g3,g12]

## Third example with the CRC polynomials of 5 bits
g1 = [1,1,0,1,0,1]
g2 = [1,0,1,0,0,1]
g3 = [1,0,1,1,1,1]
g4 = [1,1,0,0,0,1]
g5 = [1,0,0,1,1,1]
g6 = [1,0,1,1,0,1]
g7 = [1,1,0,0,1,1]
g8 = [1,1,1,1,1,1]
g9 = [1,1,1,0,0,1]
g10 = [1,0,1,0,1,1]
g11 = [1,1,1,1,0,1]
```

```
g12 = [1,1,1,0,1,1]
g13 = [1,0,0,1,0,1]
g14 = [1,0,0,0,1,1]
g15 = [1,0,0,0,0,1]
g16 = [1,1,0,1,1,1]
glst_5 = [g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,g11,g12,g13,g14,g15,g16]

w_lst_pure = []
w_lst_coded = []

for g in glst:
    print(np.poly1d(g[::-1]))
    weights = w_finder(code_gen(g,k))
    w_lst_pure.append(weights)
    print(weights)
    print("min weight w* = ",find_min_weight(weights),'\n')
print('And now the coded scheme:\n')

for g in glst:
    print(np.poly1d(g[::-1]))
    weights = w_finder_coded(g_C,n,g,k)
    w_lst_coded.append(weights)
    print(w_finder_coded(g_C,n,g,k),'\n')
    print("min weight w* = ",find_min_weight(weights))

full_plot_pure(w_lst_pure,k)
full_plot(t,w_lst_coded,n)

w_lst_pure = []
w_lst_coded = []

for g in glst2:
    print(np.poly1d(g[::-1]))
    weights = w_finder(code_gen(g,k))
    w_lst_pure.append(weights)
    print(weights)
    print("min weight w* = ",find_min_weight(weights),'\n')
print('And now the coded scheme:\n')

for g in glst2:
    print(np.poly1d(g[::-1]))
    weights = w_finder_coded(g_C,n,g,k)
    w_lst_coded.append(weights)
    print(w_finder_coded(g_C,n,g,k),'\n')
    print("min weight w* = ",find_min_weight(weights))

full_plot_pure(w_lst_pure,k)
full_plot(t,w_lst_coded,n)

w_lst_pure = []
```

```
w_lst_coded = []

for g in glst_5:
    print(np.poly1d(g[::-1]))
    weights = w_finder(code_gen(g,k))
    w_lst_pure.append(weights)
    print(weights)
    print("min weight w* = ",find_min_weight(weights),'\n')
print('And now the coded scheme:\n')

for g in glst_5:
    print(np.poly1d(g[::-1]))
    weights = w_finder_coded(g_C,n,g,k)
    w_lst_coded.append(weights)
    print(w_finder_coded(g_C,n,g,k),'\n')
    print("min weight w* = ",find_min_weight(weights))

full_plot_pure(w_lst_pure,k)
full_plot(t,w_lst_coded,n)



#########################

### BCH code 2 ####
n,k,t = 31,16,3
g_C = [1,1,1,1,0,1,0,1,1,1,1,1,0,0,0,1]


### The CRC polynomials of 10 bits compared to each other
g1 = [1,1,1,0,1,1,0,0,1,0,1]
g2 = [1,0,1,1,1,0,0,0,1,0,1]
g3 = [1,1,0,0,1,1,1,0,1,0,1]
g4 = [1,0,0,1,1,0,0,0,0,1,1]
glst_10 = create_crc_list(10)
glst = [g1,g2,g3,g4]


for g in glst_10:
    print(np.poly1d(g[::-1]))
    weights = w_finder(code_gen(g,k))
    print(weights)
    print("min weight w* = ",find_min_weight(weights),'\n')
print('And now the coded scheme:\n')

for g in glst_10:
    weights = w_finder_coded(g_C,n,g,k)
    if find_min_weight(weights) >= 13:
        print(np.poly1d(g[::-1]))
        print(weights,'\n')
```

```
        print("min weight w* = ",find_min_weight(weights))


########################################

### Comparison of 4 CRC polynomials of 10 bits

w_lst_pure = []
w_lst_coded = []

for g in glst:
    print(np.poly1d(g[::-1]))
    weights = w_finder(code_gen(g,k))
    print(weights)
    print("min weight w* = ",find_min_weight(weights),'\n')
    w_lst_pure.append(weights)

print('And now the coded scheme:\n')

for g in glst:
    print(np.poly1d(g[::-1]))
    weights = w_finder_coded(g_C,n,g,k)
    print(weights)
    print("min weight w* = ",find_min_weight(weights),'\n')
    w_lst_coded.append(weights)


full_plot_pure(w_lst_pure,k)
full_plot(t,w_lst_coded,n)
```