# Power Decoding

**Going Beyond the Limit**

Kennie Fischer

Aalborg University

Mathematics - Master Thesis

Spring 2019

**AALBORG UNIVERSITY**
STUDENT REPORT

**Mathematics - 10th semester**
Skjernvej 4A
9220 Aalborg Øst
http://www.math.aau.dk

**Title:** Power Decoding

**Subtitle** Going Beyond the Limit

**Theme:** Master Thesis

**Project period:** Spring semester 2019

**Project group:** 5.213a

**Author:** Kennie Fischer

**Supervisors:**
Ignacio P. Cascudo
Oliver W. Gnilke

**Pages:** 52

**Date of Completion:** 7th June 2019

**Abstract:**

This thesis examines two decoding methods for error correcting Reed-Solomon codes: Gao decoding and Power Decoding. We show that Gao decoding are able to correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. If the method returns a codeword this must a codeword within distance $\lfloor \frac{d-1}{2} \rfloor$ to the received message, and otherwise the method will declare failure.
Power decoding is an extension of Gao code, and for this reason it can also correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. In addition to this it can some sometimes correct more errors than this. We show examples of this as well as give some bounds on when it should not be expected to correct errors. This is also supported by simulations.
The thesis also contains a section on how to solve MgLFSR problem which is need for Power decoding.

# Preface

This master's thesis is made during the $4^{\text{th}}$ semester of the Master program in Mathematics at Aalborg University, Spring 2019. The aim of this project is to examine Power decoding based on Gao decodning. The understanding of this project implies knowledge of abstract algebra and coding theory.

The thesis contains various proofs and examples. An proof end with ■, and examples end with a ☽. The reason for including a moon in examples is the balance of examples being more grounded compared to rest of the thesis.

The reader should be warned that this the project may include jokes of various degree of wittiness. If one does not like the inclusion of jokes in serious mathematical work, one should skip such jokes.

Throughout this project, some mathematical mistakes, errors, typos and invalid arguments are included. These are in fact not true mistakes, since they are purposely put there to keep the reader critical and sceptical. For instance several times polynomials are included in inequalities. Here the reader should be sceptical and realise that a polynomial cannot be smaller than a number, but the degree of said polynomial can.

Lastly, I want to thank my supervisors Ignacio Cascudo Pueyo and Oliver Wilhelm Gnilke for guidance and advise.

Aalborg University, 7th June 2019

Kennie Fischer

kfisch14@student.aau.dk

# Danish Summary

Dette speciale omhandler Power decoding, som er et emne inden for kodningsteori. Idéen bag Power decoding er at genskabe et tættest liggende kodeord ud fra en meddelelse der ligger mere end en halv minimum afstand. Den metode vi vil benytte er en udvidelse af en ellers kendt dekoding, Gao dekodning, som benyttes til Reed-Solomonkoder $R(q, n, k)$.

Gao dekodning går ud på at finde en så kaldt minimal løsning til

$$\begin{aligned} \lambda R &\equiv \psi \mod G \\ \deg \psi &< \frac{n+k}{2} \end{aligned}$$

hvor $R$ og $G$ er kendte, og $\psi$ og $\lambda$ er ukendte. En metode til at løse dette problem på er baseret på Euklids udvidede algoritme. Ud fra den fundende løsning vil dekoderen enten erklære fejl eller returne en meddelelse. Det vises desuden, at hvis der opnås en meddelelse, så vil denne meddelelse ligge i en afstand på maksimalt $\lfloor \frac{d-1}{2} \rfloor$ til et kodeord. Afsnittet runder af med et forslag om, at gradsbetigelsen $\deg \psi < \frac{n+k}{2}$ ændres til $\deg \psi \leq \deg \lambda + (k-1)$. Dette vil resultere i, at det vil være muligt at korrigere fejl uden for en halv minimum afstand.

I afsnit 3 introduceres Power decoding. Her fastlægger man et $l$ underlagt til $l(k-1) < n$. Herefter finder man en minimal løsning til

$$\begin{aligned} \lambda R^{(t)} &\equiv \psi^{(t)} \mod G \\ \deg \psi^{(t)} &\leq \lambda + t(k-1) \end{aligned} \quad \text{for } t = 1, \ldots, l,$$

hvor $G$ og $R^{(t)}$ er kendte, og $\lambda$ og $\psi^{(t)}$ er ukendte. Ligesom Gao dekodning vurderes det ud fra læsningen, om der erklæres fejl eller returneres en meddelelse. Det viser sig dog, at hvis der returneres en meddelelse, så vil det tilhørende kodeord være af tætteste afstand fra den modtaget besked. I flere tilfælde vil det være muligt at korrigere flere fejl end en halv minimum afstand, men andre gange ville dette ikke være muligt. Der sættes øvregrænser hvornår dette ikke er mulig.

At finde den minimale løsning er et specialtilfælde af et MgLFSR problem. I afsnit 4 vises det hvordan man løser sådan nogle problemer. Kort sagt, så gøres det ved at indse at løsninger uden gradsbetingelsen udgør et frit modul. Til dette modul opskrives en basis bestående af en matrix, der efterfølgende udføres såkaldte række reduktion, indtil den er på en såkaldt svag Popov form. Ud fra en matrix på svag Popov form ses en løsning let.

Specialet rundes af med afsnit 5. Her udføres der simuleringer på forskelige Reed-solomonkoder, hvoraf det tyder på, at ved at gøre $l$ større vil det være muligt at med større sandsynlighed at rette flere fejl. Ulempen ved større $l$ er dog, at dette vil gøre problemet mere kompleks, og derved ved det tage længere tid at løse, hvilket simuerleringerne også understøtter.

# Contents

**Danish Summary** **ii**

**1 Introduction** **1**

**2 Gao decoding** **2**

2.1 Gao algorithm . . . . . 2

2.2 Extended Euclidean algorithm . . . . . 4

2.3 Solving the linearised key equation . . . . . 5

2.4 Error-correcting . . . . . 9

2.5 Modification to the degree constriction . . . . . 13

**3 Power Decoding** **17**

3.1 Power syndrome decoding . . . . . 21

3.2 The probability of error and failure . . . . . 25

**4 Multi-sequence linear feedback shift-register synthesis** **35**

4.1 Application to Power decoding . . . . . 42

**5 Simulations** **45**

**Bibliography** **48**

**A Code for simulations** **49**

# 1 Introduction

Reliable communication is an important factor of modern society. Each day, tons of data is send over the internet and other networks so information can be shared, and people can communicate with each other. A problem with these networks is that sometimes mistakes and errors happen, and data might get lost or changed. The mathematical field of coding theory is an important tool to combat these problems.

It has been established that one practical way of sending data is through Reed-Solomon codes. One of these reasons is that it is an MDS code meaning it can restore the maximal amount of errors compared to other linear codes of same length and dimension. MDS codes with length $n$, dimension $k$ and minimum distance $d = n - k + 1$ are able to restore $\varepsilon$ errors, where $\varepsilon < \frac{d}{2}$. Going beyond that seems impossible since the received message may not have an uniquely closest codeword. However if the luck is with us, the received message can have the original codeword as a closest codeword. This open up for it might possible to correct these more than $\frac{d}{2}$ errors if the errors occurs in some special entries.

One way of doing so is Power decoding, which this thesis will examine. The idea behind this is to power the entries of a received message $r = c + e$. This is useful because powered entries of $e$ is zero if and only if the original entries of $e$ i zero. This gives us more information to work with which is an indication of the possibility of correcting more than $\frac{d}{2}$ errors. Throughout this thesis, we will examine questions such as: How does Power decoding work, and how sure are we on whether it can correct more errors than $\frac{d}{2}$?

In section 2, we will introduce Gao decoding, which gives a better understanding of Power decoding since Power decoding is a extension of Gao decoding. In Section 3 we will introduce Power decoding as well as examine some of it properties. We will also see a bound of how well it works when more than $\frac{d}{2}$ errors happen. In Section 4, we will examine MgLFSR problems, which is needed to be solved in Power decoding. Finally, we will show some simulations in Section 5 that demonstrates the theory of Power decoding.

# 2 Gao decoding

This Section will introduce and explain Gao decoding. This was introduced algorithmicly in (Gao 2002), however we will suggest a alternative way of explaining the concept for a better connection to Power decoding explained in (Nielsen 2013b). We will also make use of the extended Euclidean algorithm, where the results are from (von zur Gathen and Gerhard 2013)

Before introducing Gao decoding, we will explain the setup. Let $\mathbb{F}_q$ be the field of $q$ elements. Fix distinct elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$. The Reed-Solomon code $RS(q, n, k)$, where $k < n \leq q$, is given by the set

$$C = \{(f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) \mid f \in \mathbb{F}_q[x], \deg(f) < k\}.$$

The message $m = (m_1, \ldots, m_k) \in \mathbb{F}_q^k$ is said to be encoded as $c \in \mathbb{C}$ given by the polynomial $f = m_1 x^{k-1} + \cdots + m_{k-1} x + m_k$. Thus there is a one to one correspondence between messages $m \in \mathbb{F}_q^k$, codewords $c \in \mathbb{C}$ and polynomial $f \in \mathbb{F}_q[x]$ of degree less than $k$.

Consider a codeword $c = (c_1, \ldots, c_n)$ obtained from some polynomial $f \in \mathbb{F}_q[x]$. Also consider a received message

$$r = (r_1, \ldots, r_n) = (f(\alpha_1), \ldots, f(\alpha_n)) + (e_1, \ldots, e_n)$$

where $e = (e_1, \ldots, e_n)$ is called the error. From this define $I = \{i \mid e_i \neq 0\}$ and $\varepsilon = |I|$.

Define $G = \prod_{i=1}^{n}(x - \alpha_i)$. Let $R$ be the polynomial obtained by Lagrange interpolation that satisfies $R(\alpha_i) = r_i$ and $\deg(R) < n$. Lastly, define the error locator polynomial as $\Lambda = \prod_{i \in I}(x - \alpha_i)$.

A problem in coding theory is how one should restore $c$ given only $r$. For MDS codes such as Reed-Solomon codes, this is possible when $\varepsilon < \frac{d}{2}$, where $d = n - k + 1$. One way of restoring the codeword is through Gao decoding.

## 2.1 Gao algorithm

In terms of error-correcting, we will in practice only have the received message $r$ and not the pairs $c$ and $e$. This means that we should consider them unknowns, as well as $I$ and $\Lambda$ should be considered unknown. However, $G$ is easy to determine for a Reed-Solomon code, and $R$ is easily computable for a received message $r$. The goal of error-correcting is to determine $c$. An approach to this would be to set up conditions which $f$, and hereby $c$, has to satisfy. Then hopefully we will be able to show that $f$ is the only solution to said conditions. Our first method, we will present, is the Gao decoding, which uses the Key equation seen in Proposition 2.2.

**Lemma 2.1**
Let $\mathbb{F}_q$ be a field, $f, g \in \mathbb{F}_q[x]$, and $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$. Define $G = \prod_{i=1}^{n}(x - \alpha_i)$. Then

$$f \equiv g \mod G \quad \Leftrightarrow \quad f(\alpha_i) = g(\alpha_i) \quad \text{for } i = 1, \ldots, n$$

*Proof.* If $f \equiv g \mod G$ then $f = g + kG$ for some polynomial $k \in \mathbb{F}_q[x]$. Since $\alpha_i$ is a root of $G$, we have

$$f(\alpha_i) = g(\alpha_i) + k(\alpha_i)G(\alpha_i) = g(\alpha_i) + 0 = g(\alpha_i)$$

On the other hand, if $f(\alpha_i) = g(\alpha_i)$, then $h := f - g$ has the roots $\alpha_1, \ldots, \alpha_n$. Therefore $h = 0$ or $h$ must have all $(x - \alpha_i)$ as factors, meaning $G$ is a factor of $h$. Thus $f - g = h = kG$ for some $k \in \mathbb{F}_q[x]$, implying $f \equiv g \mod G$. ■

**Proposition 2.2** (Key equation)
Using the notation of the section

$$\Lambda R \equiv \Lambda f \mod G$$

*Proof.* Due to Lemma 2.1, we only have to determine that the two polynomials have the same evaluation at $\alpha_1, \ldots, \alpha_n$. If $i \in I$, then $\alpha_i$ is a root of $\Lambda$, implying both sides to be zero. If $i \notin I$ then $e_i = 0$ and thus $R(\alpha_i) = r_i = f(\alpha_i) + e_i = f(\alpha_i)$. ■

A problem with the key equation is that it is not linear. For this reason we will linearise so it becomes

$$\lambda R \equiv \psi \mod G. \tag{1}$$

In the linearised version we ignore the algebraic structure of $\psi = \lambda f$. Also we substitute $\Lambda$ by $\lambda$ to emphasise that a solution to (1) does not need to have the form of an error locator polynomial.

So, we want to find a solution to (1) which we denote $(\lambda, \psi)$. This solution will then hopefully be $(\Lambda, f\Lambda)$. However there are infinite many solutions to 1. For instance if $(\lambda, \psi)$ is a solution, so is $(p\lambda, p\psi)$ for any polynomial $p \in \mathbb{F}[x]$. Therefore we do some constrictions on our solutions. First of all, we will use the constriction $\deg \psi < \deg \frac{n+k}{2}$. Note that when $\varepsilon < \frac{d}{2}$, we have $\deg f\Lambda < k - 1\frac{n-k+1}{2} < \frac{n+k}{2}$, meaning $(\Lambda, f\Lambda)$ still can be a solution. Furthermore when solving the key equation, we will search for a solution where $\lambda$ is monic and of smallest possible degree. We will call such a solution a minimal solution.

When a minimal solution $(\lambda, \psi)$ is found, we will determine whether it has the desired algebraic structure, i.e. $\psi = \lambda\hat{f}$ for some polynomial $\hat{f}$. If it does we should return $\hat{f}$, otherwise we should declare failure.

So to summarise the Gao method, we do the following for a received message $r = (r_1, \ldots, r_n)$ and a predetermined $G = \prod_{i=1}^{n}(x - \alpha_i)$

**Step 1:** Compute the Lagrangian polynomial $R$ of degree less than $n$ satisfying

$$R(\alpha_i) = r_i \quad \text{for } i = 1, \ldots, n$$

**Step 2:** Find a minimal solution $(\lambda, \psi)$ to

$$\lambda R \equiv \psi \mod G$$
$$\deg \psi < \frac{n+k}{2}$$

**Step 3:** Divide $\psi$ by $\lambda$ to obtain $\rho$ and $\hat{f}$ such that

$$\psi = \lambda\hat{f} + \rho. \tag{2}$$

If $\rho = 0$ return $\hat{f}$ and $\deg \hat{f} < k$, otherwise declare failure.

The procedure above will return a polynomial $\hat{f}$ if it return anything at all. In practice one might prefer it returning a message $\hat{m}$. However, there is one to one correspondence between messages $\hat{m} \in \mathbb{F}_q^k$, codewords $c \in C$ and polynomials $f \in \mathbb{F}_q[x]$ of degree less than $k$. Therefore for our view, we do not care which of these it returns because it is easy to compute the others. Throughout this thesis, we will change the output consistently inconsistently depending on what is best in the given context.

## 2.2 Extended Euclidean algorithm

One way of solving (1) is through extended Euclidean algorithm. Original the algorithm was used to find $u, v \in \mathbb{Z}$ for $x, y \in \mathbb{Z}$ such that

$$ux + vy = \gcd(x, y).$$

However the algorithm can easily be generalised to any Euclidean domain such as polynomial rings. We will for now state the algorithm for polynomials, and later we will comment on how to use it for solve (1).

**Algorithm 1** Extended Euclidean algorithm

**Input:** Polynomials $p_0, p_1 \in \mathbb{F}[x]$, where $\deg p_1 \leq p_0$

**Output:** Polynomials $u, v, p \in \mathbb{F}[x]$ such that $\gcd(p_0, p_1) = p = up_0 + vp_1$

Set $i = 0$, $u_0 = v_1 = 1$ and $u_1 = v_0 = 1$.

**while** $p_{i+1} \neq 0$ **do**

Set $i = i + 1$

Divide $p_{i-1}$ by $p_i$ to find polynomials $p_{i+1}$ and $q_i$ such that

$$p_{i-1} = q_i p_i + p_{i+1}.$$

Set

$$u_{i+1} = u_{i-1} - q_i u_i, \quad v_{i+1} = v_{i-1} - q_i v_i \tag{3}$$

**return** $u_i$, $v_i$ and $p_i$

Note that Algorithm 1 eventually terminates since otherwise the sequence $\{\deg(p_i)\}_{i \in N}$ would be a strictly decreasing sequence of natural numbers. Therefore, let us say that it ends in the $m^{\text{th}}$ iteration, i.e. $p_{m+1} = 0$. By induction, we show that

$$p_i = u_i p_0 + v_i p_1. \tag{4}$$

for $i = 0, \ldots, m+1$. Clearly it is true for $i = 0, 1$. For $i + 1 > 1$, we have

$$\begin{aligned} p_{i+1} &= p_{i-1} - q_i p_i \\ &= u_{i-1} p_0 + v_{i-1} p_1 - q_i u_i p_0 - q_i v_i p_1 \\ &= (u_{i-1} - q_i u_i) p_0 + (v_{i-1} - q_i v_i) p_1 \\ &= u_{i+1} p_0 + v_{i+1} p_1 \end{aligned}$$

We will now argue that it returns the correct polynomials, i.e. $\gcd(p_0, p_1) = u_m p_0 + v_m p_1$. This follows from

$$\gcd(p_{i-1}, p_i) = \gcd(q_i p_i + p_{i+1}, p_i) = \gcd(p_{i+1}, p_i) = \gcd(p_i, p_{i+1})$$

for $i = 1, \ldots, m$. Using this iteratively, we end up with

$$\gcd(p_0, p_1) = \gcd(p_m, p_{m+1}) = \gcd(p_m, 0) = p_m = u_m p_0 + v_m p_1$$

## 2.3 Solving the linearised key equation

We will now explain how we can use the extended Euclidean algorithm to solve (1). For this we use Algorithm 1 with input $p_0 = G$ and $p_1 = R$ such that we achieve $uG + vR = \gcd(G, R)$.

Then it immediately follows that $vR \equiv \gcd(G, R) \mod G$, meaning that $(v, \gcd(G, R))$ is a solution to (1). Note that the greatest common divisor is only unique up to an unit, but regardless we will have a solution. However the solution found by the extended Euclidean algorithm is not necessarily the minimal solution. To find the minimal solution, we will note one observation of importance. Under each iteration of the extended Euclidean algorithm we will compute $u_i, v_i, p_i$ such that $u_i G + v_i R = p_i$. This means that for each $i$ we have a solution $(v_i, p_i)$. Thus we can run the extended Euclidean algorithm and save each pair $(v_i, p_i)$. Then we can take the minimal of the computed pairs and hope that it is the minimal solution globally. We will later see in Proposition 2.4 that this is in fact a minimal solution.

An alternative and more elegant method than just listing the pairs $(v_i, p_i)$ would be to note that the sequence of $v_i$ is increasing and the sequence of $p_i$ is decreasing. This is important when dealing with the degree constriction $\deg p_i < \frac{n+k}{2}$. Since at some point achieve a solution satisfying the degree constriction, there exists a smallest $j$ such that for $i \geq j$ we have $\deg p_i < \frac{n+k}{2}$. Our goal is now to show that the minimal solution is $(v_j, p_j)$, which would imply that the minimal solution is among the ones gained from the extended Euclidean algorithm.

**Lemma 2.3**

Run the extended Euclidean algorithm with input $p_0$ and $p_1$. Then

$$\deg v_i = \sum_{j=1}^{i-1} \deg q_j = \deg p_0 - \deg p_{i-1} \quad \text{for } i = 1 \ldots, m.$$

*Proof.* We want to prove

$$\begin{aligned} \deg v_i &= \sum_{j=1}^{i-1} \deg q_j = \deg p_0 - \deg p_{i-1} \qquad \text{for } i = 1, \ldots, m \\ \deg v_{i-1} &< \deg v_i \end{aligned} \tag{5}$$

by induction. For $i = 1$ we have $v_1 = 1$, $v_0 = 0$ and $\sum_{j=1}^{0} \deg q_j$ is an empty sum, thus $\deg v_1 = 0 = \deg p_0 - \deg p_0$, and $\deg v_0 = -\infty < 0 = \deg v_1$. So assume that (5) is true for $i$ and we want to prove for $i + 1 \leq m + 1$. Since $q_i \neq 0$ for $i \leq m$, we have

$$\deg v_{i+1} = \deg(v_{i-1} - q_i v_i) = \deg q_i + \deg v_i > \deg v_i$$

Also

$$\deg v_{i+1} = \deg q_i + \deg v_i = \deg q_i + \sum_{j=1}^{i-1} \deg q_j = \sum_{j=1}^{(i+1)-1} \deg q_j$$

Finally we have that

$$\begin{aligned}\sum_{j=1}^{i} \deg q_j &= \sum_{j=1}^{i-1} \deg q_j + \deg q_i \\ &= \deg p_0 - \deg p_{i-1} + \deg q_i \\ &= \deg p_0 - \deg(p_{i+1} + q_i p_i) + \deg q_i \\ &= \deg p_0 - \deg q_i - \deg p_i + \deg q_i \\ &= \deg p_0 - \deg p_i\end{aligned}$$

■

**Proposition 2.4**

Suppose

$$up_0 + vp_1 = p$$

for some $u, v, p, p_0, p_1 \in \mathbb{F}[x]$ where $v \neq 0$ and

$$\deg p + \deg v < \deg p_0 =: n$$

Run the extended Euclidean algorithm with $p_0$ and $p_1$ as input to obtain $(u_i, v_i, p_i)$ for $i = 0, \ldots, m$. Define $j$ such that $p_j \leq p < p_{j-1}$. Then there exist some $\alpha \in \mathbb{F}[x]$ such that

$$u = \alpha u_j, \quad v = \alpha v_j, \quad p = \alpha p_j$$

*Proof.* Assume for contradiction that $u_j v \neq u v_j$. Note that we have

$$\begin{bmatrix} u_j & v_j \\ u & v \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{bmatrix} p_j \\ p \end{bmatrix}$$

The matrix above is invertible since $u_j v \neq u v_j$, meaning

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \frac{1}{u_j v - v_j u} \begin{bmatrix} v & -v_j \\ -u & u_j \end{bmatrix} \begin{bmatrix} p_j \\ p \end{bmatrix}$$

and therefore

$$p_0 = \frac{v p_j - v_j p}{u_j v - v_j u} \tag{6}$$

We get by Lemma 2.3 that

$$\begin{aligned}\deg(p_j v - v_j p) &\leq \max\{\deg p_j + \deg v, \deg v_j + \deg p\} \\ &\leq \max\{\deg p + \deg v, n - \deg p_{j-1} + \deg p\} \\ &< \max\{n, \deg p_{j-1} + n - \deg p_j\} = n\end{aligned}$$

Meaning that the polynomial of the right hand side of (6) has degree strictly less than $n$. This is however a contradiction with $p_0$ having degree $n$, meaning $u_j v = u v_j$.

Let us now prove $\gcd(u_i, v_i) = 1$ for $0 \leq i \leq m$. From (3), we have

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{bmatrix}.$$

Using this iteratively gives

$$\begin{bmatrix} u_i & v_i \\ u_{1+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -q_{i-1} \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 \\ 1 & -q_1 \end{bmatrix} \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \end{bmatrix}.$$

Due to the initial values $u_0 = v_1 = 1$ and $u_1 = v_0 = 0$, the last matrix of above is the identity matrix. Thus taking the determinant gives

$$\det \begin{bmatrix} u_i & v_i \\ u_{1+1} & v_{i+1} \end{bmatrix} = \prod_{j=1}^{i} \det \begin{bmatrix} 0 & 1 \\ 1 & -q_j \end{bmatrix} = \prod_{j=1}^{i} (-1) = (-1)^i \tag{7}$$

Thus we have $u_i v_{i+1} + u_{i+1} v_i = (-1)^i$ implying $\gcd(u_i, v_i) = 1$.

Since $v_j$ divides $u_j v = u v_j$ and $\gcd(u_j, v_j) = 1$, we have that $v_j$ divides $v$. Therefore $v = \alpha v_j$ for some $\alpha \in \mathbb{F}[x]$. Since $v \neq 0$ we have $v_j \neq 0$, and thus we can divide $u v_j = u_j v = u_j \alpha v_j$ by $v_j$ to get $u = \alpha u_j$. Finally

$$p = u p_0 + v p_1 = \alpha u_j p_0 + \alpha v_j p_1 = \alpha (u_j p_0 + v_j p_1) = \alpha p_j$$

■

It follows from Proposition 2.4 that the minimal solution must have have the form $(\alpha v_i, \alpha p_i)$ for some $i \in \mathbb{N}$ and $\alpha \in \mathbb{F}[x]$. This implies however that $(v_i, p_i)$ is a solution satisfying the degree constriction, meaning $\alpha \in \mathbb{F}$. Thus the minimal solution is $(v_j, p_j)$ where $j$ is the smallest such that $\deg p_j < \frac{n+k}{2}$. We will therefore use a modified extended Euclidean algorithm with $G$ and $R$ as input, but instead of stopping when $p_{i+1} = 0$ we will stop when $\deg p_{i+1} < d_0 := \frac{n+k}{2}$. This algorithm will return $(u_j, v_j, p_j)$ where the minimal solution is $(v_j, p_j)$.

**Algorithm 2** Modified extended Euclidean algorithm

**Input:** Polynomials $p_0, p_1 \in \mathbb{F}[x]$ and $d_0 \in \mathbb{N}$

**Output:** Polynomials $u, v, p \in \mathbb{F}[x]$ such that $p = up_0 + vp_1$ and $\deg p < d_0$

Set $i = 0$, $u_0 = v_1 = 1$ and $u_1 = v_0 = 0$.

**while** $\deg p_{i+1} \geq d_0$ **do**

Set $i = i + 1$

Divide $p_{i-1}$ by $p_i$ to find polynomials $p_{i+1}$ and $q_i$ such that

$$p_{i-1} = q_i p_i + p_{i+1}.$$

Set

$$u_{i+1} = u_{i-1} - q_i u_i, \quad v_{i+1} = v_{i-1} - q_i v_i$$

**return** $u_{i+1}$, $v_{i+1}$ and $p_{i+1}$

## 2.4 Error-correcting

So far, we have seen how Gao decoding works. However we want also to show that it can correct errors. For MDS codes, such as Reed-Solomon codes, we are able correct up to $\frac{d-1}{2}$ errors. The question is now whether the Gao method can do it. Spoiler warning: it does, as we will see in upcoming Theorem 2.7. For proving this we will need a couple of lemmas.

**Lemma 2.5**

Let $p_0, p_1 \in \mathbb{F}[x]$ be nonzero and distinct. Let Algorithm 1 end in the $m^{\text{th}}$ iteration, then

$$u_{m+1} = (-1)^{m+1} \frac{p_1}{p_m}, \quad v_{m+1} = (-1)^m \frac{p_0}{p_m}$$

*Proof.* From (4), we have

$$\begin{bmatrix} p_i \\ p_{i+1} \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}.$$

Note that the matrix in the equation above is invertible due to (7), which also implies

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix}^{-1} \begin{bmatrix} p_i \\ p_{i+1} \end{bmatrix} = (-1)^i \begin{bmatrix} v_{i+1} & -v_i \\ -u_{i+1} & u_i \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \end{bmatrix}.$$

Using the stopping criterion $p_{m+1} = 0$ we get by setting $i = m$ that

$$p_0 = (-1)^m v_{m+1} p_m, \quad p_1 = (-1)^m (-u_{m+1} p_m)$$

implying the lemma. ■

Before stating the next lemma, we will explain some notation. First of all if we run Algorithm 1 and 2 with input $g_0$ and $g_1$ instead of $p_0$ and $p_1$, we will refer the polynomials computed in each iteration as $g_i$.

**Lemma 2.6**
Let $g_0 = wp_0 + \varepsilon_0$ and $g_1 = wp_1 + \varepsilon_1$ with $\gcd(p_0, p_1) = 1$ and

$$\deg p_i \leq t, \text{ and } \deg \varepsilon_i \leq l \text{ for } i = 1, 2. \tag{8}$$

Suppose $d_0$ satisfies

$$\deg w \geq d_0 > l + t$$

Apply the Algorithm 2 to $g_0$, $g_1$ and $d_0$. Suppose that it returns $(u, v, g)$ such that

$$ug_0 + vg_1 = g.$$

Then

$$u = -ap_1, \quad v = ap_0$$

for some nonzero $a \in \mathbb{F}$.

*Proof.* Let us first apply Algorithm 1 to $p_0$ and $p_1$, meaning we find

$$p_{i-1} = q_i p_i + p_{i+1}, \quad \deg p_{i+1} < \deg p_i \text{ for } i = 1, \ldots m$$

where $p_{m+1} = 0$ and $p_i \neq 0$ for $i = 0, \ldots m$, and

$$\begin{aligned} u_0 = 1, \quad u_1 = 0, \quad u_{i+1} = u_{i-1} - q_i u_i \quad &\text{for } i = 1, \ldots m \\ v_0 = 1, \quad v_1 = 0, \quad v_{i+1} = v_{i-1} - q_i v_i \quad &\text{for } i = 1, \ldots m \end{aligned} \tag{9}$$

whereas

$$p_i = u_i p_0 + v_i p_1 \text{ for } i = 1, \ldots m + 1$$

and

$$\deg u_i \leq \deg p_1 \leq t \text{ and } \deg u_i \leq \deg p_1 \leq t \text{ for } i = 1, \ldots m + 1.$$

Now we will do computations of the extended Euclidean algorithm to $g_0$ and $g_1$, but instead of updating $u_i$ and $v_i$ with respect to $p_i$, we will use the one computed by $g_i$ given in (9). We define

$$g_i = u_i g_0 + v_i g_1 \quad \text{for } i = 2, \ldots m + 1. \tag{10}$$

Note that the above is also valid for $i = 0, 1$. We will now show that these definitions agree with the dividing procedure of the extended Euclidean algorithm. By combining (10) and

(9), we get for $i = 1, \ldots, m$ that

$$\begin{aligned} g_{i-1} &= u_{i-1}g_0 + v_{i-1}g_1 \\ &= (u_{i+1} + q_i u_i)g_0 + (v_{i+1} + q_i v_i)g_1 \\ &= (u_{i+1}g_0 + v_{i+1}g_1) + q_i(u_i g_0 + v_i g_1) \\ &= g_{i+1} + q_i g_i. \end{aligned}$$

We will now show that the degrees of $g_i$ has the proper degrees. So for $i = 2, \ldots m+1$

$$\begin{aligned} g_i &= u_i g_0 + v_i g_1 \\ &= u_i(wp_0 + \varepsilon_0) + v_i(wp_1 + \varepsilon_1) \\ &= w(u_i p_0 + v_i p_1) + (u_i\varepsilon_0 + v_i\varepsilon) \\ &= wp_i + (u_i\varepsilon + v_i\varepsilon). \end{aligned}$$

By the assumption of the lemma, we have $\deg(u_i\varepsilon_0 + v_i\varepsilon) \leq l + t < d_0 = \deg w$. So for $i = 1, \ldots, m$

$$\deg g_i = \deg w + \deg p_i \geq \deg w \geq d_0$$

and for $i = m + 1$

$$\deg g_{m+1} = \deg(u_i\varepsilon + v_i\varepsilon) \leq l + t < d_0$$

since $p_{m+1} = 0$. This means the sequence of $\deg g_i = \deg w + \deg p_i$ is strictly decreasing, since the sequence of $\deg(p_i)$ is. Thus due to the uniqueness of division with remainder, we have that $g_{i-1} = q_i g_i + g_{i+1}$ if one would try to divide $g_{i-1}$ with $g_i$. In other words, the $q_i$ gained by Algorithm 1 with input $p_0$ and $p_1$, is the same $q_i$ gained by Algorithm 2 with input $g_0$ and $g_1$. Thus also $u_i$ and $v_i$ are the same regardless of algorithm.

Since $\deg(g_i) \geq d_0$ for $\leq< m$, and $\deg(g_{m+1}) < d_0$, Algorithm 2 will stop in the $m^{\text{th}}$ iteration, meaning

$$g = u_{m+1}g_0 + v_{m+1}g_1$$

where $u_{m+1} = u$ and $v_{m+1} = v$. Now it follows from Lemma 2.5 that $u_{m+1} = (-1)^{m+1}p_1/p_m$ and $u_{m+1} = (-1)^m p_0/p_m$. Since $p_m = \gcd(p_0, p_1) = 1$, we have now proven the lemma bu setting $a = (-1)^m$. ■

**Theorem 2.7**

Let $r \in \mathbb{F}_q^n$ be a received message. If $r$ and some codeword $c \in C$ is within distance $(d-1)/2$, the Gao decoding will return $c$. Otherwise it will declare failure.

*Proof.* Let $r = (r_1, \ldots, r_n)$ be a received message with distance $\varepsilon \leq (d-1)/2$ from the codeword $c = (c_1, \ldots, c_n)$ made from the polynomial $f \in \mathbb{F}[x]$. Recall the error locator polynomial

$$\Lambda = \prod_{i \in I}(x - \alpha)$$

Define $\Lambda_0 = \frac{G}{\Lambda} = \prod_{i \notin I}(x - \alpha_i)$, and define $\tilde{\Lambda}$ to be the Lagrangian polynomial satisfying

$$\tilde{\Lambda}(\alpha_i) = \frac{r_i - c_i}{\Lambda_0(\alpha_i)} \quad \text{for } i \in I$$

Note that this is possible since $\Lambda_0(\alpha_i) = 0$ if and only if $i \notin I$.

The first thing we want to show is that $\gcd(\Lambda, \tilde{\Lambda}) = 1$. Since $\Lambda$ is a product of first degree, it is equivalent with stating that they share a root. So assume for contradiction that there exist $j \in I$ such that $\Lambda(\alpha_j) = \tilde{\Lambda}(\alpha_j) = 0$. This would mean that $\tilde{\Lambda}(\alpha_j) = (r_j - c_j)/\Lambda(\alpha_j) = 0$, implying $r_j = c_j$. This can however not be the case since $j \in I$, meaning that we have shown $\gcd(\Lambda, \tilde{\Lambda})$.

Next we want to prove that

$$R = \Lambda_0 \tilde{\Lambda} + f.$$

Since both sides have degree strictly less than $n$, we will show that they agree on $\alpha_i$ for $i = 1, \ldots, n$. If $i \in I$ then

$$\Lambda_0(\alpha_i)\tilde{\Lambda}(\alpha_i) + f(\alpha_i) = \Lambda_0(\alpha_i)\frac{r_i - c_i}{\Lambda_0(\alpha_i)} + c_i = r_i = R(\alpha_i).$$

On the other hand if $i \notin I$ then $\alpha_i$ is a root of $\Lambda_0$, implying $f(\alpha_i) = c_i = r_i = R(\alpha_i)$.

Set $d_0 = (n + k)/2$. Then

$$\deg(\Lambda_0) = n - \varepsilon \geq n - \frac{d-1}{2} = n - \frac{n - k + 1 - 1}{2} = \frac{n+k}{2} = d_0$$

and

$$d_0 = \frac{d-1}{2} + k > \varepsilon + k - 1.$$

We will now use Lemma 2.6. For a better understanding, we will highlight the notation. The left hand sides are notations of the lemma while the right hand sides are the notations of this proof:

$$\begin{array}{lll} g_0 = G & p_0 = \Lambda & \varepsilon_0 = 0 \\ g_1 = R & p_1 = \tilde{\Lambda} & \varepsilon_1 = f \\ w = \Lambda_0 & t = \varepsilon & l = k - 1 \quad d_0 = \dfrac{n+k}{2} \end{array}$$

We will therefore by Lemma 2.6 end up with

$$u = -a\tilde{\Lambda}, \quad v = a\Lambda$$

in Algorithm 2. Thus

$$g = uG + vR = -a\tilde{\Lambda}\Lambda\Lambda_0 + a\Lambda(\Lambda_0\tilde{\Lambda} + f) = a\Lambda f = vf$$

This will imply when we divide $g$ by $v$, we will get $f$ with no remainder. Hence the algorithm will return $f$.

On the other hand, assume the algorithm returns a polynomial $\hat{f}$. We will now prove that the induced codeword is within distance $(d-1)/2$ of the input $r$.

Since the method returns a polynomial the remainder $\rho$ is 0 meaning $g = v\hat{f}$, and thus $uG = v(\hat{f} - R)$. By construction of $G$, it has roots $\alpha_i$ for $i = 1, \ldots, n$, and therefore

$$v(\alpha_i)(\hat{f}(\alpha_i) - R(\alpha_i)) = 0 \quad \text{for } i = 1, \ldots, n. \tag{11}$$

Due to the stopping criteria of Algorithm 2 we have

$$\deg(v) = \deg(g) - \deg(f) \leq \deg(g) < \frac{n+k}{2} = n - \frac{d-1}{2}.$$

So at most $n - \frac{d-1}{2}$ of the roots of (11) can come from $v$ since $v \neq 0$. This means at least $n - \frac{d-1}{2}$ values of $i$ satisfies $(f - R)(\alpha_i) = 0$, and thus also $f(\alpha_i) = R(\alpha_i)$. Thus there are at most $(d-1)/2$ values of $i$ satisfying $f(\alpha_i) \neq R(\alpha_i)$, meaning the received message is within distance $(d-1)/2$ of a codeword.

■

Before closing this section off, we will emphasise the importance of Theorem 2.7. Recall that $\varepsilon$ is the number of entries where errors have happened. If $\varepsilon \leq \frac{d-1}{2}$, then we able to correct the errors. On the other hand, if $\varepsilon > \frac{d-1}{2}$ then two things can happen. A received message is within distance $\frac{d-1}{2}$ to a codeword different from the original, and in this case, we will decode the wrong codeword. Hopefully, this will not happen, and we will not be in distance $\frac{d-1}{2}$ to any codeword. If this is the case, we will declare failure.

## 2.5 Modification to the degree constriction

To prepare us for what to come, we will modify the constriction on the degrees of the powered key equation. Instead of using $\deg\psi < \frac{n+k}{2}$, we will use $\deg\psi < \deg\lambda + (k-1)$. Therefore we have to distinguish between our original problem of finding a minimal solution to

$$\begin{aligned} \lambda R &\equiv \psi \mod G \\ \deg\psi &< \frac{n+k}{2} \end{aligned} \tag{12}$$

and our new problem of finding a minimal solution to

$$\begin{aligned} \lambda R &\equiv \psi \mod G \\ \deg\psi &\leq \deg\lambda + (k-1). \end{aligned} \tag{13}$$

The minimal solution to (12) will often be the same as the minimal solution to (13), but not always. Due to Proposition 2.4, the solution to (13) must be gained from an iteration of the extended Euclidean algorithm. We will see in the following proposition how the two solutions are related.

**Proposition 2.8**
Run the Euclidean algorithm on $p_0 = G$ and $p_1 = R$ to get $(p_i, u_i, v_i)$ for $i = 0, \ldots, m$. Let $(v_j, p_j)$ be a minimal solution to (12) then $(v_j, p_j)$ or $(v_{j+1}, p_{j+1})$ is a minimal solution to (13).

*Proof.* We will prove this by showing that $(v_{j-1}, p_{j-1})$ does not satisfy the degree constriction in (13), but $(v_{j+1}, p_{j+1})$ does. So since $v_j$ is a minimal solution, $(v_{j-1}, p_{j-1})$ do not satisfy (12) and thus we get by Lemma 2.3 that

$$\begin{aligned}
\deg v_{j-1} + (k-1) &= \deg G - \deg p_{j-2} + k - 1 \\
&< n - \deg p_{j-1} + k - 1 \\
&\leq n - \frac{n+k}{2} + k - 1 \\
&< \frac{n+k}{2} \\
&\leq p_{j-1}
\end{aligned}$$

Similarly, $(v_j, p_j)$ and $(v_{j+1}, p_{j+1})$ must satisfy (12), and thus

$$\begin{aligned}
\deg v_{j+1} + (k-1) &= \deg G - \deg p_j + k - 1 \\
&\geq n - \frac{n+k}{2} + k \\
&= \frac{n+k}{2} \\
&> p_{j+1}
\end{aligned}$$

■

Recall that in Gao decoding we search for a minimal solution to (12) and hope that this is $(\Lambda, \Lambda f)$. We will now suggest doing the same but using (13) instead of (12). In practice if one uses the extended Euclidean algorithm, the approach would be almost same except one should stop the computations when $\deg p_i \leq v_i + (k-1)$.

**Proposition 2.9**
Gao decoding with degree constriction (13) succeeds if Gao decoding with degree constriction (12) succeeds.

*Proof.* Since Gao decoding succeeds the minimal solution to (12) must have the form $(\Lambda, \Lambda f)$ where $\deg f < k$. Let us say that this solution is obtained by the extended Euclidean algorithm as $(v_i, p_i) = (\Lambda, \Lambda f)$. By Proposition 2.8, the minimal solution to (13) is either $(v_i, p_i)$ or $(v_{i+1}, p_{i+1})$. This must necessary be $(v_i, p_i) = (\Lambda, \Lambda f)$ since it satisfies the degree constriction and $\deg v_i < \deg v_{i+1}$. ■

So whenever $\varepsilon < \frac{d}{2}$, the two methods both succeed. However when $\varepsilon \geq \frac{d}{2}$, the original method fails according to Theorem 2.7, but using instead (13) will sometimes succeed. One of these instances can be seen in the following example.

**Example 2.10.** Consider the Reed Solomon code with $q = 13$, $n = 11$, $k = 4$ and evaluation points $a_i = i$ for $i = 1, \ldots 11$. This means $d = 11 - 4 + 1 = 8$. Assume we have the message $m = (0, 0, 0, 0)$ which get encoded to the codeword $c = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$. Assume we receive the message $r = (5, 9, 0, 10, 0, 0, 10, 0, 0, 0, 0)$. This means that $\varepsilon = 4 > \lfloor \frac{7}{2} \rfloor = \lfloor \frac{d-1}{2} \rfloor$ and Gao decoding should not be able to decode it with degree constriction (12). Firstly let us compute

$$G = \prod_{i=1}^{n}(x - i) = x^{11} + 12x^{10} + x^9 + 12x^8 + x^7 + 12x^6 + x^5 + 12x^4 + x^3 + 12x^2 + x + 12$$

$$R = 8x^{10} + 3x^9 + 6x^8 + 7x^7 + x^6 + 7x^5 + 7x^4 + 6x^2 + 10x + 2$$

We will now run the extended Euclidean algorithm on $p_0 = G$ and $p_1 = R$. The result of this can be seen in table 1

| $i$ | $q_i$ | $p_i$ | $u_i$ | $v_i$ |
|---|---|---|---|---|
| 0 | | $x^{11} + 12x^{10} + x^9 + 12x^8 + x^7 + 12x^6$ $+x^5 + 12x^4 + x^3 + 12x^2 + x + 12$ | 1 | 0 |
| 1 | $5x + 11$ | $8x^{10} + 3x^9 + 6x^8 + 7x^7 + x^6$ $+7x^5 + 7x^4 + 6x^2 + 10x + 2$ | 0 | 1 |
| 2 | $7x + 5$ | $3x^9 + 2x^8 + 10x^7 + 5x^6$ $+6x^5 + 10x^3 + 11x + 3$ | 1 | 8x + 2 |
| 3 | $4x + 7$ | $4x^8 + 12x^6 + 3x^5 + 2x^4$ $+2x^3 + 7x^2 + 12x$ | $6x + 8$ | $9x^2 + 11x + 4$ |
| 4 | $4x$ | $x^7 + 3x^5 + 4x^4 + 7x^3$ $+7x^2 + 5x + 3$ | $2x^2 + 4x + 10$ | $3x^3 + 10x^2 + 6x$ |
| 5 | | 0 | $5x^3 + 10x^2 + 5x + 8$ | $x^4 + 12x^3 + 11x^2 + 11x + 4$ |

Table 1: The computations of the extended Euclidean algorithm with input $p_0 = G$ and $p_1 = R$

Remember the two degree constriction from (12) and (13). For a clear view on the degrees we will refer to table 2. Here we see that $i = 4$ is the the smallest $i$ such that $\deg p_i < \frac{n+k}{2} =$

$\frac{11+4}{2} = 7.5$ meaning $(v_4, p_4) = (x^7 + 3x^5 + 4x^4 + 7x^3 + 7x^2 + 6x + 3, 3x^3 + 10x^2 + 6x)$ is the minimal solution to (12). However $i = 5$ is the smallest $i$ such that $\deg p_i \leq v_i + (k-1)$ meaning $(v_5, p_5) = (x^4 + 12x^3 + 11x^2 + 11x + 4, 0)$ is the minimal solution to (13).

| $i$ | $\deg p_i$ | $\deg v_i + (k-1)$ |
|---|---|---|
| 0 | 11 | 2 |
| 1 | 10 | 3 |
| 2 | 9 | 4 |
| 3 | 8 | 5 |
| 4 | 7 | 6 |
| 5 | $-\infty$ | 7 |

Table 2: Degrees of various polynomials

Next step is to determine whether $(v_4, p_4)$ has the form of $(\lambda, \lambda f)$. Therefore we divides $p_4$ by $v_4$ such that we get

$$(x^7+3x^5+4x^4+7x^3+7x^2+5x+3) = (x^3+10x^2+6x)(9x^4+9x^3+5x^2+10x+11)+(6x^2+4x+3)$$

Since the remainder is nonzero we should declare failure. This fits well with Theorem 2.7, since $\varepsilon \geq \frac{d}{2}$.

If we instead use the solution $(v_5, p_5)$ and divide $p_5 = 0$ by $v_5$ we will clearly get $\hat{f} = 0$ and a remainder of 0. This corresponds to the message $\hat{m} = (0, 0, 0, 0)$ meaning we have successfully restored the original message. This is remarkable since $\varepsilon > \lfloor \frac{d-1}{2} \rfloor$, and the original Gao decoding could not correct this many errors.

Ↄ

Example 2.10 is remarkable since $\varepsilon > \lfloor \frac{d-1}{2} \rfloor$ and the original Gao decoding could not correct this many errors.Therefore it seems better to use (13) compared (12) when finding a minimal solution. In fact in the next section about Power decoding, we will see a generalisation of (13) that make us able to correct more errors.

# 3 Power Decoding

In this section we will introduce and explain the decoding method known as Power decoding. This will be based on version in (Nielsen 2013b), where it is called Power Gao. However some results are taken from other sources, and when these come relevant, we will refer to said sources.

As mentioned Power decoding can be seen as an extension to Gao decoding. Hence we will use much of the notation from Section 2. In addition we will define $R^{(t)}$ for a given $t$ as the Lagrangian polynomial satisfying $R^{(t)}(\alpha_i) = r_i^t$. Recall $G = \prod_{i=1}^{n}(x - \alpha_i)$ and $\Lambda = \prod_{i \in I}(x - \alpha_i)$. Lastly we need $l$ subjected to $l(k-1) < n$. Our first result is the Powered Key equation.

**Proposition 3.1** (Powered Key equation)
Using the notation of the section

$$\Lambda R^{(t)} \equiv \Lambda f^t \mod G$$

*Proof.* Due to Lemma 2.1, we only have to determine that the two polynomials have the same evaluation at $\alpha_1, \ldots, \alpha_n$. If $i \in I$, then $\alpha_i$ is a root of $\Lambda$, implying both sides to be zero. If $i \notin I$ then $e_i = 0$ and thus $R^{(t)}(\alpha_i) = r_i^t = (f(\alpha_i) + e_i)^t = f(\alpha_i)^t$. ■

The concept and approach of Power decoding is similar to Gao decoding. Instead of the key equation from Proposition 2.2, we will use the powered key equation from Proposition 3.1. So just as previously we will linearise the equation such that we get

$$\lambda R^{(t)} \equiv \psi^{(t)} \mod G \qquad \text{for } t = 1, \ldots, l \tag{14}$$

where we have substituted $\Lambda$ by $\lambda$ and $\Lambda f^t$ by $\psi^{(t)}$. A solution to 14 must therefore have the form $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$. This would hopefully be the solution $= (\Lambda, \Lambda f, \ldots, \Lambda f^t)$. We will once again have a restrictions on the degrees of the polynomials. We will use the constriction $\deg \psi^{(t)} \leq \deg \lambda + t(k-1)$, since

$$\deg \psi^{(t)} = \deg \Lambda f^t = \deg \lambda + t \deg f \leq \deg \lambda + t(k-1).$$

Note the similarities between this constriction and (13). Once again when solving (14), we will search for a minimal solution. Recall that $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$ is minimal if $\lambda$ is monic and the degree of $\lambda$ is minimal of all possible solution. We will later see in section 4 how to find such a solution.

When a minimal solution is found, then we should determine whether it has the algebraic form of $(\Lambda, \Lambda f, \ldots, \Lambda f^t)$, i.e. there exists a polynomial $\hat{f}$ such that

$$\psi^{(t)} = \lambda \hat{f}^t \quad \text{for } t = 1, \ldots, l \tag{15}$$

Note that that if $\hat{f}$ exists, it must have the form $\hat{f} = \frac{\psi^{(1)}}{\lambda}$. If (15) is true return $\hat{f}$, if not we will declare failure.

Let us summarise the method of Power decoding.

**Step 1:** Compute the Lagrangian polynomials $R^{(t)}$ for $t = 1, \ldots, l$ of degree less than $n$ satisfying

$$R^{(t)}(\alpha_i) = r_i \quad \text{for } i = 1, \ldots, n$$

**Step 2:** Find a minimal solution $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$ to

$$\begin{aligned} \lambda R^{(t)} &\equiv \psi^{(t)} \mod G \\ \deg \psi^{(t)} &\leq \lambda + t(k-1) \end{aligned} \quad \text{for } t = 1, \ldots, l. \tag{16}$$

**Step 3:** Divide $\psi^{(1)}$ by $\lambda$ to obtain $\rho$ and $\hat{f}$ such that

$$\psi = \lambda \hat{f} + \rho. \tag{17}$$

If $\rho \neq 0$ declare failure.

**Step 4** Determine whether $\lambda \hat{f} \equiv \psi^{(t)} \mod G$ for $t = 2, \ldots, l$. If so return $\hat{f}$, and if not declare failure.

So there is three different out comes. Hopefully we will have that $\hat{f} = f$, and if this is the case Power decoding have succeeded. If either $\hat{f} \neq f$ or we declare failure, the method fails. The question is now when does the method succeed and when does it fail? The first result we have is that this question only depends on $e$ an not on our original codeword.

**Proposition 3.2**
Let $c \in C$ and $e \in \mathbb{F}_q^n$. Power decoding fails for $r = c + e$ if and only if it fails for $e$.

*Proof.* For each $t$, let $R_e^{(t)}$ be the Lagrangian polynomial satisfying $R_e^{(t)}(\alpha_i) = e_i^t$. Let $R^{(t)}$ be the Lagrange polynomial defined in the usual way, i.e. $R^{(t)}(\alpha_i) = r_i^t$. For simplicity let $R = R^{(1)}$ and $R_e = R_e^{(1)}$.

The idea of this proof is to create a one to one correspondence between the solutions of the linearised power key equation for $e$ and for $r$. So consider a solution $(\lambda, \psi_e^{(1)}, \ldots, \psi_e^{(l)})$ to the linearised power key equation for $e$. Per definition this means

$$\lambda R_e^{(t)} \equiv \psi_e^{(t)} \mod G \quad \text{where } \deg \psi_e^{(t)} \leq \lambda + t(k-1)$$

Note that $R^t \equiv R^{(t)} \mod G$ and $R_e^t \equiv R_e^{(t)} \mod G$ due to Lemma 2.1. Let $f$ be the polynomial corresponding to $c$, then $R = f + R_e$ since $r_i = c_i + e_i = f(\alpha_i) + e_i$. Define $\psi_e^{(0)} = \lambda$ and notice that for $t = 0$ we have $\deg \psi_e^t < \lambda + t(k-1) + 1$.

It now follows from the binomial formula

$$\begin{aligned}\lambda R^{(t)} &\equiv \lambda R^t \equiv (f + R_e)^t \\ &\equiv \lambda \sum_{s=0}^{t} \binom{t}{s} f^s R_e^{t-s} \equiv \sum_{s=0}^{t} \binom{t}{s} f^s \lambda R_e^{(t-s)} \\ &\equiv \sum_{s=0}^{t} \binom{t}{s} f^s \psi_e^{(t-s)} \mod G\end{aligned}$$

Each term of the sum on the right hand side have degree

$$s \deg f + \deg \psi_e^{t-s} \leq s(k-1) + \deg \lambda + (t-s)(k-1) = \deg \lambda + t(k-1).$$

Thus the sum will also have degree lower than $\deg \lambda + t(k-1)$. This means we have found

$$(\lambda, \sum_{s=0}^{1} \binom{1}{s} f^s \psi_e^{(1-s)}, \ldots, \sum_{s=0}^{t} \binom{l}{s} f^s \psi_e^{(l-s)})$$

to be a solution to the linearised power key equation for $r$.

On the other hand if $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$ is a solution to the linearised power key equation for $r$, one can use similar arguments to show that

$$((\lambda, \sum_{s=0}^{1} (-1)^s \binom{1}{s} f^s \psi^{(1-s)}, \ldots, \sum_{s=0}^{t} (-1)^s \binom{l}{s} f^s \psi^{(l-s)})$$

is a solution to the linearised power key equation for $e$.

This means that there exist a one-to-one correspondence between solutions of the linearised power key equations for $r$ and $e$. Furthermore the two solutions will have the same $\lambda$. Thus if it possible to find a solution of the linearised power key equation for $r$ with $\deg \lambda < \deg \Lambda$ it is possible to find a solution of the linearised power key equation for $c$ with $\deg \lambda < \deg \Lambda$. This will imply that both will fail or both will success.

■

Let $l_0 < l_1$. If $(\lambda, \psi^{(1)}, \ldots, \psi^{(l_1)})$ is a the solution to (14) with $l = l_1$ then $(\lambda, \psi^{(1)}, \ldots, \psi^{(l_0)})$ is a solution to (14) with $l = l_0$. The reason for this is that solving (14) with $l = l_0$ is a subproblem of solving it with $l = l_1$. However a minimal solution for $l = l_1$ may not necessarily induce a minimal solution when $l = l_0$. Contrasting to this $(\lambda, \psi^{(1)}, \ldots, \psi^{(l_0)})$ is minimal solution for $l = l_0$ and $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$ is solution for $l = l_1$, then $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$ must be a minimal solution for $l = l_1$. The reasons for this is that if it was not the minimal,

it would have induced a solution for $l = l_0$ where the degree of $\lambda$ would be smaller than the original. Therefore if Power decoding with $l = l_0$ succeeds then Power decoding with $l = l_1$ succeeds, where $l_0 < l_1$. Recall that in Section 2.5 a new degree constriction to the Gao decoding was suggested. This corresponds with Power decoding in the case $l = 1$. Also in that section, we showed that when $\varepsilon < \frac{d}{2}$ then the Gao method succeeds. This means that Power decoding will succeed if $\varepsilon < \frac{d}{2}$ for any $l$.

**Example 3.3.** Consider the Reed Solomon code $C$ over the field $\mathbb{F}_7$ with $n = 5$, $k = 2$ and $\alpha_i = i$ for $i = 1, \ldots, 5$. This gives that

$$G = \prod_{i=1}^{5}(x - \alpha_i) = x^5 + 6x^4 + x^3 + 6x^2 + x + 6. \tag{18}$$

Let us say that we have the message $m = (2, 3)$ meaning we have the polynomial $f = 2x + 3$. Thus if we encode $m$ we get the codeword

$$c = (f(1), f(2), \ldots, f(5)) = (5, 0, 2, 4, 6)$$

Note that $d = 5 - 2 + 1 = 3$, meaning we are sure that we can correct up to $\lfloor \frac{d-1}{2} \rfloor = 1$ error. However let us try to go beyond that, and say 2 errors have happened, namely $e = (0, 0, 6, 5, 0)$. This means that we receive the message

$$r = c + e = (5, 0, 1, 2, 6).$$

So let us firstly decide an $l$ subjected to $l(k-1) < n$, where we choose $l = 3$. So our first step of decoding is to interpolate polynomials such that $R(i)^t = r_i^t$ for $i = 1, \ldots, 5$ and $t = 1, 2$. Doing so we will get

$$\begin{aligned} R^{(1)} &= 3x^4 + 4x^3 + 2x^2 + 3 \\ R^{(2)} &= 3x^4 + x^3 + 2x^2 + x + 4 \end{aligned} \tag{19}$$

Next we will write up the the powered key equation. Remember that we have the degree constriction $\deg \psi^{(t)} \leq \deg \lambda - t(k-1)$ meaning we have to solve

$$\begin{aligned} \lambda R^{(t)} &\equiv \psi^{(t)} \mod G \\ \deg \psi^{(t)} &\leq \deg \lambda - t \end{aligned} \quad \text{for } t = 1, 2$$

where $\lambda$ and $\psi^{(t)}$ are unknowns, and $R^{(t)}$ and $G$ are given as in (18) and (19). We will later see in Example 4.12 how to get the minimal solution. But for now let us claim that it is given by

$$\begin{aligned} \lambda &= x^2 + 5 \\ \psi^{(1)} &= 2x^3 + 3x^2 + 3x + 1 \\ \psi^{(2)} &= 4x^4 + 5x^3 + x^2 + 4x + 3 \end{aligned}$$

We now divide $\psi^{(1)}$ by $\lambda$ to obtain $\hat{f}$ and $\rho$ such that $\psi^{(1)} = \lambda\hat{f} + \rho$ . This will imply that $\hat{f} = 2x3$ and $\rho = 0$. Since $\rho = 0$ we have to test whether $\lambda\hat{f}^t = \psi^{(t)}$. This is in fact the case, and thus we return the message $(2, 9, 5)$. Hence we have successfully restored the message.

However when dealing with messages beyond minimum distance this method depends on where the error happens. For instance if the error is $(1, 0, 0, 2, 0, 0, 0, 9)$, we will get the minimal solution to be

$$\begin{aligned}\lambda &= x^2 + 2x \\ \psi^{(1)} &= 5x^3 + 2x^2 + x + 6 \\ \psi^{(2)} &= x^4 + x^3 + x^2 + 3x + 1.\end{aligned}$$

If we try to obtain to obtain $\hat{f}$ and $\rho$, we see

$$5x^3 + 2x^2 + x + 6 = (5x + 6)(x^2 + 2x) + (3x + 6)$$

meaning $\rho \neq 0$ and thus we should declare failure.

$\mathfrak{D}$

## 3.1 Power syndrome decoding

Power decoding was originally defined in a different way than the one we present. The original version was suggest in (Schmidt et al. 2007), and we will refer to this method as Power syndrome. The notation in there however differs a lot from ours, why our explanation will instead be based on (Nielsen 2013b).

Before introducing Power syndrome decoding, we will need to define the reciprocal polynomial of $p = a_0 + a_1x + \cdots + a_nx^n \in \mathbb{F}_q[x]$ as $\overline{p} = a_n + a_{n-1}x + \cdots + a_0x^n$. In Lemma 3.4, certain rules regarding reciprocal polynomials is shown.

**Lemma 3.4**
Let $p, q \in \mathbb{F}[x]$ and $n = \deg p$, $m = \deg q$, $l = \deg(p + q)$ then

1. $\overline{p} = x^n p(x^{-1})$
2. $\overline{(pq)} = (\overline{p})(\overline{q})$
3. $\overline{(p + q)} = x^{l-n}\overline{p} + x^{l-m}\overline{q}$
4. $\overline{x^k f} = \overline{f}$ for any $k \in \mathbb{N}$.
5. The first nonzero coefficient of $p$ is the $\mathrm{k}^{\mathrm{th}}$ if and only if $x^k\overline{\overline{p}} = p$

*Proof.* For the first claim let $p = a_0 + a_1 x + \cdots + a_n x^n$, then

$$x^n p(x^{-1}) = x^n(a_0 + a_1 x^{-1} + \cdots + a_n x^{-n}) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n = \overline{p}$$

For the second claim

$$\overline{(pq)} = x^{n+m}(pq)(x^{-1}) = x^n p(x^{-1}) x^m q(x^{-1}) = (\overline{p})(\overline{q})$$

For the third claim

$$\overline{p+q} = x^l(p(x^{-1}) + q(x^{-1})) = x^l p(x^{-1}) + x^l q(x^{-1}) = x^{l-n}\overline{f} + x^{l-m}\overline{g}$$

For the fourth claim combine the second claim with

$$\overline{x^k} = x^k(x^{-1})^k = 1$$

For the fourth claim let $p = a_k x^k + \ldots + a_n x^n$ where $a_k \neq 0$. Then $\overline{p} = a_n + \cdots + a_k x^{n-k}$ and thus $\overline{\overline{p}} = a_k + \ldots a_n^{n-k}$ since $a_k \neq 0$. Multiplying by $x^k$ gives $x^k\overline{\overline{p}} = p$.

On the hand if $x^k\overline{\overline{p}} = p$, we know $x^k$ is a factor of $p$ and thus $a_0 = \cdots = a_{k-1} = 0$. ■

**Remark 3.5.** In case where $l < \max(n, m)$, the terms on the right hand side of claim 3 might not be polynomial. Therefore one should be careful with using this rule.

In power syndrome we have the same setup as previously. Furthermore, we assume 0 is not amoung the evaluation points $\alpha_1, \ldots \alpha_n$.

For $t = 1, \ldots, l$ we will define the syndrome polynomial by

$$S^{(t)} = \sum_{i=1}^{n} \frac{r_i^t \sigma_i}{1 - x\alpha_i} \mod x^{n-t(k-1)+1}$$

where $\sigma_i = \prod_{j \neq i}(\alpha_i - \alpha_j)^{-1}$. Note that $1 - x\alpha_i$ has inverse modulo $x^{n-t(k-1)+1}$ since it is of first order and does not divide $x^{n-t(k-1)+1}$.

Recall that $\Lambda = \prod_{i \in I}(x - \alpha_i)$ meaning

$$\overline{\Lambda} = x^{\varepsilon} \prod_{i \in I}(x^{-1} - \alpha_i) = \prod_{i \in I} x(x^{-1} - \alpha_i) = \prod_{i \in I}(1 - \alpha_i x)$$

Note that this means the roots of $\Lambda$ are $\alpha_i^{-1}$ where $i \in I$

We will also make use of the so called powered error evaluator polynomial $\Omega^{(t)}$ which we will not define explicitly. The only need for it is to state

$$\overline{\Lambda} S^{(t)} \equiv \Omega^{(t)} \mod x^{n-t(k-1)+1} \text{ for } t = 1 \ldots, l$$

accordingly to (Nielsen 2013b). Treating $\Lambda$ and $\Omega^{(t)}$ as unknowns by substituting them by $\lambda$ and $\omega^{(t)}$ we get the powered powered syndrome equation

$$\overline{\lambda} S^{(t)} \equiv \omega^{(t)} \mod x^{n-t(k-1)+1} \text{ for } t = 1 \ldots, l \tag{20}$$

The idea behind Power syndrome decoding is to solve (20), and thus get $\lambda$. Afterwards one can find $\hat{I}$ as the $i$ such that $\lambda(\alpha_i) = 0$. Once again the solution $(\overline{\lambda}, \omega^{(1)}, \ldots, \omega^{(l)})$ we are searching for is a minimal one.

Unlike the Power Gao method, the Power Syndrome method only gives the entries where the error have happened but not an estimated codeword. Another difference is that it never declares failure. This means that we succeed if and only if $\hat{I} = I$ and fails if and only if $\hat{I} \neq I$. We will therefore summarise Power syndrome decoding in the following.

**Step 1:** Compute the polynomials $S^{(t)}$ for $t = 1, \ldots, l$ as

$$S^{(t)} = \sum_{i=1}^{n} \frac{r_i^t \sigma_i}{1 - x\alpha_i} \mod x^{n-t(k-1)+1}$$

**Step 2:** Find a minimal solution $(\lambda, \omega^{(1)}, \ldots, \omega^{(l)})$ to

$$\begin{aligned} \overline{\lambda} S^{(t)} &\equiv \omega^{(t)} \mod x^{n-t(k-1)+1} \\ \deg \omega^{(t)} &< \deg \lambda \end{aligned} \qquad \text{for } t = 1, \ldots, l. \tag{21}$$

**Step 3:** Return $\hat{I}$ such that

$$\lambda(\alpha_i) = 0 \quad \text{for } i \in \hat{I}.$$

In (Nielsen 2013b), it was suggested that Power decoding and Power syndrome decoding is equivalent in the sense that the both fail and succeed for the same received codewords. However it seems that the proof of this statement in (Nielsen 2013b) is a bit insufficient since it does not take into account the possibilities $\deg \omega^{(t)} < \deg \lambda - 1$ and $\deg R^{(t)} < n - 1$. Let us instead state a proposition which we are able to proof.

**Proposition 3.6**
Let $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$ be a minimal solution to (16) implying the existence of $\omega^{(t)}$ such that $\lambda R^{(t)} - \omega^{(t)} G = \psi^{(t)}$. Assume $\deg R^{(t)} = n - 1$ and $\deg \omega^{(t)} = \deg \lambda - 1$. Then Power syndrome fails if Power decoding fails.

*Proof.* Since $R^{(t)}$ is made to interpolate $r_i^t$ at $\alpha_i$, it is given by $R^{(t)} = \sum_{i=1}^{n} r_i^t \sigma_i \prod_{j \neq i}(x - \alpha_j)$, where $\sigma_i = \prod_{j \neq i}(\alpha_i - \alpha_j)^{-1}$.

Note that

$$\overline{G} = x^n \prod_{j=1}^{n}(x^{-1} - \alpha_j) = \prod_{j=1}^{n} x(x^{-1} - \alpha_j) = \prod_{j=1}^{n}(1 - \alpha_j x).$$

This means that it has inverse modulo $x^{n-t(k-1)+1}$ since it is product of first degree polynomial which does not divide $x^{n-t(k-1)+1}$. Thus we see $\overline{R^{(t)}}(\overline{G})^{-1} \equiv S^t \mod x^{n-t(k-1)+1}$ since

$$\frac{\overline{R^{(t)}}}{\overline{G}} \equiv \sum_{i=1}^{n} \frac{r_i^t \sigma_i \prod_{j \neq i}(1 - \alpha_j x)}{\prod_{j=1}^{n}(1 - \alpha_j x)} \equiv \sum_{i=1}^{n} \frac{r_i^t \sigma_i}{1 - x\alpha_i} \equiv S^t \mod x^{n-t(k-1)+1}$$

We have by Lemma 3.4 that

$$\begin{aligned}
\lambda R^{(t)} - \omega^{(t)} G &= \psi^{(t)} \\
\overline{(\lambda R^{(t)} - \omega^{(t)} G)} &= \overline{\psi^{(t)}} \\
x^{\deg \psi^{(t)} - (\deg \lambda + n - 1)} \overline{\lambda} \overline{R^{(t)}} - x^{\deg \psi^{(t)} - (\deg \lambda - 1 + n)} \overline{\omega^{(t)}} \overline{G} &= \overline{\psi^{(t)}} \\
\overline{\lambda} \overline{R^{(t)}} - \overline{\omega^{(t)}} \overline{G} &= \overline{\psi^{(t)}} x^{\deg \lambda - 1 + n - \deg \psi^{(t)}}
\end{aligned}$$

Since $\deg \psi^{(t)} \leq \deg \lambda + t(k-1)$ we have

$$\overline{\lambda} \overline{R^{(t)}} - \overline{\omega^{(t)}} \overline{G} \equiv 0 \mod x^{n - t(k-1) - 1}$$

Rearranging and dividing both sides by $\overline{G}$ gives

$$\overline{\lambda} S^{(t)} \equiv \overline{\omega^{(t)}} \mod x^{n - t(k-1) - 1}$$

since $S^{(t)} = \overline{R^{(t)}} \overline{G}^{-1}$. This means that we have found a solution to the powered syndrome equation since $\deg \overline{\omega^{(t)}} \leq \deg \omega^{(t)} \leq \deg \lambda - 1$. Since Power decoding fails we have that $\deg \lambda \leq \deg \Lambda$. Therefore this same $\lambda$ can make Power syndrome fail. ■

**Remark 3.7.** The assumption in Proposition 3.6 that $\deg R^{(t)} = n - 1$ and $\deg \omega^{(t)} = \deg \lambda - 1$ can be possible. The reason for this is that $R^{(t)}$ is a Lagrangian polynomial satisfying $n$ conditions, and

$$\begin{aligned}
\deg \omega^{(t)} &\leq \deg(\psi^{(t)} - \lambda R^{(t)}) - \deg(G) \\
&\leq \max(\deg \psi^{(t)}, \deg(\lambda R^{(t)})) - \deg(G) \\
&\leq \max(\deg \lambda + t(k-1), \deg(\lambda) + n - 1) - n \\
&= \deg \lambda + n - 1 - n \\
&= \deg \lambda - 1
\end{aligned}$$

Let us for now assume the condition $\deg R^{(t)} = n - 1$ and $\deg \omega^{(t)} = \deg \lambda - 1$. This means by Propositon 3.6 that Power decoding is to prefer compared with Power syndrome. There are a few reasons for this. First of all Power decoding allows the evaluation point $\alpha_i = 0$, which in some application is useful. More importantly Power decoding allows declaring failure. This means we can sometimes detect that have failed compared to just getting a wrong codeword.

Let us introduce the decoding radius $\varepsilon_{\max}$. For a decoder, the decoding radius is the biggest radius such that

- there exist a codeword $c$ and a received message $r$ where $d(c, r) = \varepsilon_{\max}$, and $r$ is decoded as $c$, and

- there does not exist a pair of a codeword $c$ and a received message $r$ where $d(c, r) > \varepsilon_{\max}$, and $r$ is decoded as $c$.

In (Schmidt et al. 2007), it was suggest that the decoding radius for Power syndrome decoding is given by $\max_{j=0,\ldots,l} \tau(j)$ where $\tau$ is given by

$$\tau(l) = \frac{l}{l+1}n - \frac{1}{2}l(k-1) - \frac{l}{l+1}. \tag{22}$$

Assuming Power decoding fails if and only if Power syndrome fails, the decoding radius for Power decoding is also $\varepsilon_{\max} = \max_{j=0,\ldots,l} \tau(j)$. An important thing to note about $\tau$ is how it behaves for different values of $n$ and $k$. For bigger values of $n$ and low values of $k$, $\tau$ have a bigger maximum, and thus $\varepsilon_{\max}$ becomes bigger. This suggest that Power decoding is best suited for lower rate Reed-Solomon codes since the rate is given by $\frac{k}{n}$.

So when $\varepsilon < \frac{d}{2}$, Power decoding will succeed, and when $\varepsilon > \varepsilon_{\max}$ Power decoding will fail. The question is now what happens when $\frac{d}{2} \leq \varepsilon \leq \varepsilon_{\max}$. The answer is that power decode will sometimes succeed and sometimes fail. We will now move into looking whether we should expect failure of success.

## 3.2 The probability of error and failure

Given the number of errors $\varepsilon$, we will define the the probability of error and the probability of failure in the following way. Chose a codeword $c \in C$ and generate an vector $e \in \mathbb{F}_q^n$ uniformly of weight $\varepsilon$. The probability of failure $P_f(\varepsilon)$ is the probability of either declaring failure or error correcting $e$ as codeword different from $c$. The probability of error $P_e(\varepsilon)$ is the probability of decoding error correcting $e$ as codeword different from $c$. Note that the probability depends only on the generation of $e$ and not $c$ due to proposition 3.2.

Since power decoding succeeds whenever $\varepsilon < \frac{d}{2}$, we have that $P_e(\varepsilon) = P_f(\varepsilon) = 0$ for $\varepsilon < \frac{d}{2}$. Likewise when $P_f(\varepsilon)$ for $\varepsilon > \varepsilon_{\max}$. We will now look intro how these probabilities behaves for $\frac{d}{2} \leq \varepsilon < \varepsilon_m ax$.

The first result we will present is from (Schmidt et al. 2007). This was proven for Power syndrome decoding, and assuming that Power decoding fails if and only if Power syndrome decoding fails we have the following proposition.

**Proposition 3.8**

Given $l = 2$, the probability $P_f$ that Power decoding fails is for $\varepsilon \geq d/2$ at most

$$\frac{q^{\varepsilon}}{(q-1)^{\varepsilon+1}} q^{3(\varepsilon - \tau(2))}$$

Next we will present a similar result for $l = 3$ suggested in (Nielsen 2013b), but firstly we need a lemma.

**Lemma 3.9**
Let $U \in \mathbb{F}_q[x]$ be of degree $n$, an consider the integers $k_1 < k_2 < k_3 < n$ such that $2k_2 \leq k_1 + k_3$. Let

$$S = \{(f_1, f_2, f_3) \mid f_1 f_3 \equiv f_2^2 \mod U,\ f_2 \text{ is monic},\ \deg f_t < k_t \text{ for } t = 1, 2, 3\}.$$

Then

$$\begin{aligned} |S| &\leq 3^{k_2-1} q^{k_2} && \text{if } k_1 + k_3 - 2 < n \\ |S| &\leq 2^{k_1+k_3-2} q^{k_1+k_2+k_3-n-1} && \text{if } k_1 + k_3 - 2 \geq n \end{aligned}$$

*Proof.* If $k_1 + k_3 - 2 < n$ then

$$2 \deg f_2 \leq \deg f_1 f_3 = \deg f_1 + \deg f_3 < K_1 - 1 + k_3 - 1 < n = \deg U$$

and thus $f_1 f_3 \mod U = f_1 f_3$ and $f_2^2 \mod U = f_1 f_2$. Hence $f_1 f_3 = f_2^2$ There is $(q^{k_2} - 1)/(q-1)$ ways of choosing $f_2$ as a monic polynomial for the following reason. There is $q^k - 1$ ways of choosing $f_2$ as a nonzero polynomial, and $q - 1$ polynomials can be made into the same monic polynomial by dividing by their leading coefficient. The polynomial $f_2$ contains at most $k_2 - 1$ factors of irreducible polynomials in $f_2$. Since $f_2^2 = f_1 f_3$ the $k_2 - 1$ factors with multiplicity 2 of $f_2$ must be divided among $f_1$ and $f_3$. Each factor with multiplicity 2 has 3 options; both can go to $f_1$, both can go to $f_3$, or one can go to each $f_1$ and $f_3$. This means that the factors of $f_2$ can be distributed among $f_1 f_3$ in $3^{k_2-1}$. Lastly we can choose the leading coefficient of $f_1$ in $(q - 1)$ ways. This means that for $k_1 - k_3 - 2 < n$

$$|S| \leq \frac{q^{k_2} - 1}{q - 1} 3^{k_2-1}(q - 1) < q^{k_2} 3^{k_2-1}$$

If $k_1 + k_3 - 2 \geq n$, once again there is $(q^{k_2} - 1)/(q - 1)$ ways of choosing $f_2$. The polynomial $f_1 f_3$ can be choosen among the set $f_2^2 + U \cdot \mathbb{F}[x]$. Since $\deg f_1 f_3 \leq k_1 + k_3 - 2$ and $2k_2 \leq k_1 + k_3$, the set can be restricted to $\{f_2^2 + U \cdot g \mid \deg g \leq k_1 + k_3 - 2 - n\}$. Therefore there is at most $q^{k_1+k_3-1-n}$ possibilities for just $f_1$. The number of possibilities for a monic $f_2$ is $(q^{k_2} - 1)/(q - 1)$. Thus the number of possibilities for $f_1 f_3$ is upper bounded by

$$q^{k_1+k_2+k_3-n-1}/(q - 1).$$

The polynomial $f_1 f_3$ must contain at most $k_1 + k_3 - 2$ irreducible polynomials. These factors can be distributed among $f_1$ and $f_3$ in $2^{k_1+k_2-2}$ ways. Lastly the leading coefficient of $f_1$ can be chosen in $(q - 1)$ ways. ■

**Proposition 3.10**

Given $l = 3$, the probability that power decoding fails is for $\varepsilon > d/2$ at most

$$\begin{cases} \left(\frac{q}{q-1}\right)^{\varepsilon} \left(\frac{3}{q}\right)^{2\varepsilon-(n-2(k-1))} q^{3(\varepsilon-\tau(2))+k-2} & \text{for } \varepsilon < \tau(2) - \frac{1}{3}k + 1 \\ \left(\frac{q}{q-1}\right)^{\varepsilon} 2^{2(2\varepsilon-d)+2(k-1)} q^{4(\varepsilon-\tau(3))-1} & \text{for } \varepsilon \geq \tau(2) - \frac{1}{3}k + 1 \end{cases}$$

where $\tau$ is given as in (22).

*Proof.* By Proposition 3.2, we assume $c = 0$ and thus $r = e$. This means that $R^{(t)}(\alpha_i) = 0$ for $i \notin I$ and thus $R^{(t)}$ must contain the factor $\Gamma = \prod_{i \notin I}(x - \alpha_i)$. Therefore $R^{(t)} = E^{(t)}\Gamma$ for some $E^{(t)}$ with $\deg E^{(t)} < \varepsilon$.

Assume that power decoding fails meaning that there exist $(\lambda, \psi^{(1)}, \psi^{(2)}, \psi^{(3)})$ such that $\lambda \neq \Lambda$, $\deg \lambda \leq \deg \Lambda$, $\deg \psi^{(t)} \leq \deg \lambda + t(k-1)$ and

$$\lambda R^{(t)} \equiv \psi^{(t)} \mod G.$$

Since $R^{(t)}$ and $G$ is divisible by $\Gamma$ so must $\psi^{(t)}$, and thus we define $\hat{\psi}^{(t)} = \frac{\psi^{(t)}}{\Gamma}$. Therefore we have

$$\lambda E^{(t)} \equiv \hat{\psi}^{(t)} \mod \Lambda.$$

Let $E$ be the the Lagrangian polynomial of degree $\deg E < \varepsilon$ satisfying $E(\alpha_i) = e_i$ for $i \in I$. Also let $\hat{\lambda}$ be the Lagrangian polynomial satisfying $\hat{\lambda}(\alpha_i) = \lambda(\alpha_i)\Gamma(\alpha_i)^{-1}$ for $i \in I$. Note that this is possible since $\Gamma(\alpha_i) = 0$ if and only if $i \notin I$. Thus we have that for $i \in I$ that

$$\lambda E^{(t)}(\alpha_i) = \lambda(\alpha_i)E^{(t)}(\alpha_i) = \lambda(\alpha_i)R^{(t)}(\alpha_i)\Gamma(\alpha_i)^{-1} = \lambda(\alpha_i)\Gamma(\alpha_i)^{-1}r_i^t = \hat{\lambda}(\alpha_i)e_i^t.$$

By Lemma 2.1, we get that $\lambda E^{(t)} \equiv \hat{\lambda}E^t \mod \Lambda$.

Let $I$ be fixed. Remember the result of Lagrange interpolation stating an equivalence between $(a_1, \ldots, a_\varepsilon)$ and polynomials of degree less than $\varepsilon$. Similar there is an equivalence between $(e_i)_{i \in I}$, where $e_i \neq 0$, and polynomials of degree less than $\varepsilon$ where no $\alpha_i$ is root for $i \in I$. The aim of this proof is to find all possible $e$ such that their might be an error, but instead of doing that we will search for all possible $E$.

So given $I$, the probability of power decoding failing is given by $T_\Lambda/(q-1)^\varepsilon$ where $T_\Lambda$ is the number of $E$ such that

$$\begin{aligned} \deg(E) < \varepsilon, \quad E(\alpha_i) \neq 0 \text{ if and only if } i \in I, \quad \exists(\hat{\lambda}, \hat{\psi}^{(1)}, \hat{\psi}^{(2)}, \hat{\psi}^{(3)}) \in \mathbb{F}_q^4[x] : \\ \hat{\lambda}E^t \equiv \hat{\psi}^{(t)} \mod \Lambda, \quad \deg \hat{\psi}^{(t)} \leq \deg \Lambda + t(k-1) - (n - \deg \Lambda) \end{aligned}$$

Note that this implies $\hat{\psi}^{(1)}\hat{\psi}^{(3)} \equiv (\hat{\psi}^{(2)})^2$. Therefore we define $\hat{T}_\lambda$ as the number of triples $(\hat{\psi}^{(1)}, \hat{\psi}^{(2)}, \hat{\psi}^{(3)}) \in \mathbb{F}_q^3$ satisfying

$$\hat{\psi}^{(1)}\hat{\psi}^{(3)} \equiv (\hat{\psi}^{(2)})^2 \mod \Lambda, \quad \deg \hat{\psi}^{(t)} \leq \deg \Lambda + t(k-1) - (n - \deg \Lambda)$$

The degree condition can also be rewritten as $\deg \hat{\psi}^{(t)} < 2\varepsilon - (n - t(k-1) - 1)$. Lastly we will use $\hat{T}_\Lambda^M$ which is the same as $\hat{T}_\Lambda$ but with the restriction $\hat{\psi}^{(2)}$ is monic.

We will now prove that $T_\Lambda \leq \hat{T}_\Lambda$. So let $E$ contribute to $T_\Lambda$ then the induced triple $(\hat{\psi}^{(1)}, \hat{\psi}^{(2)}, \hat{\psi}^{(3)})$ will contribute to $\hat{T}_\Lambda$. We will need

$$\hat{\psi}^{(1)}E \equiv \lambda E^2 \equiv \hat{\psi}^{(2)} \mod \Lambda \tag{23}$$

First assume $\gcd(\hat{\lambda}, \Lambda) = 1$. Then we will prove that no two $E$ can give the the same $(\hat{\psi}^{(1)}, \hat{\psi}^{(2)}, \hat{\psi}^{(3)})$. We have that $\gcd(E, \Lambda) = 1$ since $E(\alpha_i) = e_i \neq 0$ for $i \in I$ and $\Lambda = \prod_{i \in I}(x - \alpha_i)$. Combining this with $\gcd(\hat{\lambda}, \Lambda) = 1$ and $\hat{\lambda}E \equiv \hat{\psi}^{(1)} \mod \Lambda$, we can divide (23) by $\hat{\psi}^{(1)}$ to obtain $E \equiv \hat{\psi}^{(2)}/\hat{\psi}^{(1)} \mod \Lambda$. This means $E$ is uniquely obtained from $(\hat{\psi}^{(1)}, \hat{\psi}^{(2)})$ and thus no two $E$ contributes to the same triple in $\hat{T}_\Lambda$.

Generally if $\gcd(\hat{\lambda}, \Lambda) = g$ we can similarly show that $E \equiv \left(\frac{\hat{\psi}^{(2)}}{g}\right) / \left(\frac{\hat{\psi}^{(1)}}{g}\right) \mod \lambda/g$. This means that potentially from $q^{\deg g}$ different $E$ we can get the same triple, namely those of the form $fE_0$ where $\deg(f) < \deg g$ and $E_0 = \left(\frac{\hat{\psi}^{(2)}}{g}\right) / \left(\frac{\hat{\psi}^{(1)}}{g}\right) \mod \lambda/g$. However given $E_0$ we can find $q^{\deg g}$ triples contributing to $\hat{T}_\Lambda$ given by $(f\hat{\psi}^{(1)}/g, f\hat{\psi}^{(2)}/g, f\hat{\psi}^{(3)}/g)$ where $\deg f < \deg g$. This means that for a set of $q^{\deg g}$ pairs of $(\hat{\psi}^{(1)}, \hat{\psi}^{(2)})$ we can uniquely determine a set of $q^{\deg g}$ polynomials $E$.

Thus we have $\hat{T}_\Lambda \geq T_\Lambda$. However we want to relate $T_\Lambda$ to $\hat{T}_\Lambda^M$. So as seen earlier we can find $(\hat{\psi}^{(1)}, \hat{\psi}^{(2)}, \hat{\psi}^{(3)})$. Besides this, a nonzero $\beta \in \mathbb{F}_q$ will give the triple $(\beta\hat{\psi}^{(1)}, \beta\hat{\psi}^{(2)}, \beta\hat{\psi}^{(3)})$, which will contribute to $\hat{T}_\Lambda$. By choosing $\beta$ such that $\beta\hat{\psi}^{(2)}$ is monic, we have also found a contribution to $\hat{T}_\Lambda^M$. Thus we have shown $\hat{T}_\Lambda^M \geq T_\Lambda$.

We will now use Lemma 3.9 to determine $\hat{T}_\Lambda^M$. By using $\Lambda = U$, $\varepsilon = n$ $f_t = \hat{\psi}^{(t)}$, $k_t = 2\varepsilon - (n - t(k-1) - 1)$ for $t = 1, 2, 3$ to obtain

$$\begin{aligned} 3^{k_2-1}q^{k_2} \quad & \text{for } 2\varepsilon - (n - (k-1) - 1) + 2\varepsilon - (n - 3(k-1) - 1) - 2 < \varepsilon \\ 2^{k_1+k_3-2}q^{k_1+k_2+k_3+\varepsilon-2} \quad & \text{for } 2\varepsilon - (n - (k-1) - 1) + 2\varepsilon - (n - 3(k-1) - 1) - 2 \geq \varepsilon \end{aligned}$$

The rest of the proof is to rewriting this. So first of all note that

$$\begin{aligned} \tau(2) &= \frac{2}{3}n - \frac{1}{2}2(k-1) - \frac{2}{3} = \frac{2}{3}n - k + \frac{1}{3} \\ \tau(3) &= \frac{3}{4}n - \frac{3}{2}(k-1) - \frac{3}{4} = \frac{3}{4}n - \frac{3}{2}k - \frac{3}{4} \end{aligned}$$

This means that

$$\begin{aligned} & 2\varepsilon - (n - (k-1) - 1) + 2\varepsilon - (n - 3(k-1) - 1) - 2 < \varepsilon \\ \Leftrightarrow \quad & 3\varepsilon - 2n + 4k - 4 < 0 \end{aligned}$$

implying

$$\varepsilon < \frac{2}{3}n - \frac{4}{3}k + \frac{4}{3} = \tau(2) - \frac{1}{3}k + 1.$$

We have

$$\begin{aligned}
&3^{k_2-1}q^{k_2} \\
=&3^{2\varepsilon-(n-2(k-1)-1)-1}q^{2\varepsilon-(n-2(k-1)-1)} \\
=&3^{2\varepsilon-n+2k-2}q^{2\varepsilon-n+2k-1} \\
=&3^{2\varepsilon-n+2k-2}q^{3\varepsilon-2n+4k-3}q^{-(2\varepsilon-n+2k-2)}q^{\varepsilon} \\
=&q^{\varepsilon}\left(\frac{3}{q}\right)^{2\varepsilon-n+2k-2}q^{3\varepsilon-2n+3k-1+k-2} \\
=&q^{\varepsilon}\left(\frac{3}{q}\right)^{2\varepsilon-(n-2(k-1))}q^{3(\varepsilon-\tau(2))+k-2}
\end{aligned}$$

We have

$$\begin{aligned}
&2^{k_1+k_3-2}q^{k_1+k_2+k_3-\varepsilon-1} \\
=&2^{2(2\varepsilon-n+1)+(1+3)(k-1)}q^{3(2\varepsilon-n+1)-\varepsilon+6(k-1)-1} \\
=&2^{2(2\varepsilon-n+k-1)+2(k-1)}q^{5\varepsilon-3n-3+6k-1} \\
=&q^{\varepsilon}2^{2(2\varepsilon-d)+2(k-1)}q^{4(\varepsilon-\tau(3))-1}
\end{aligned}$$

So we have now a upper bound on the size of favourable outcomes, which we divide by the size of possible outcomes, namely $(q-1)^{\varepsilon}$, this will imply the proposition.

■

So we have now given bound on the probability of failure for $l = 2$ and $l = 3$. The question whether the method declares failure can be somewhat complex. Therefore we will now examine when Power decoding returns the wrong codeword. This is a bit simpler since Power decoding is a special type of decoder, namely it satisfies the ML certificate.

**Definition 3.11**
An decoder satisfies the maximum likelihood (ML) certificate property if a received message $r$ will either be decoded as

- a declared failure, or
- a codeword $\hat{c}$ such that

$$d(r, \hat{c}) = \min_{c \in C} d(r, c).$$

**Proposition 3.12**

Let the Power decoding return a polynomial $f$, where $(\lambda, \psi(1), \ldots, \psi^{(l)})$ is the minimal solution to the powered key equation. Then $\deg f \leq k-1$ and $\lambda$ has the form of a error locator polynomial

$$\lambda = \prod_{i \in I}(x - \alpha_i)$$

for some $I \subseteq \{1, \ldots, n\}$.

*Proof.* Firstly it follows that $\deg f \leq k-1$ from the degree constriction of the key equation since

$$\deg \lambda + t \deg f = \deg \lambda f^t = \deg \psi^{(t)} \leq \deg \lambda + t(k-1).$$

Subtracting $\deg \lambda$ and dividing by $t$ gives $\deg f \leq k-1$.

Assume that the algorithm have returned with the minimal solution $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$. Since we have not declared error, there must exist $f$ such that $\psi^{(t)} = \lambda f^t$ for $t = 1 \ldots, l$. A solution must satisfy the powered key equation meaning

$$\lambda R^{(t)} \equiv \lambda f^t \mod G \quad \text{for } t = 1, \ldots, l \tag{24}$$

where $\deg \lambda f^t \leq \deg \lambda + t(k-1)$. Factorise $\lambda = \lambda_G \lambda_0$, where $\lambda_G = \gcd(G, \lambda)$. Assume for contradiction that $\deg \lambda_0 > 0$. Since $\gcd(\lambda_0, G) = 1$, we can divide (24) by $\lambda_0$ to obtain

$$\lambda_G R^{(t)} \equiv \lambda_G f^t \mod G \quad \text{for } t = 1, \ldots, l.$$

This means that $(\lambda_G, \lambda_G f, \ldots \lambda_G f^t)$ satisfies the powered key equation. Also it must satisfy the degree constrictions since

$$\deg(\lambda_G f^t) = \deg(\lambda f^t) - \deg \lambda_0 \leq \deg \lambda + t(k-1) - \deg \lambda_0 = \deg \lambda_G + t(k-1).$$

Thus $(\lambda_G, \lambda_G f, \ldots \lambda_G f^t)$ is a solution with $\deg \lambda_G < \deg \lambda$. This is a contradiction with $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$ being a minimal solution, meaning $\lambda_0 = 1$. Thus we have that $\lambda = \gcd(\lambda, G)$. Since $G = \prod_{i=1}^{n}(x - \alpha_i)$, its only factors have form of an error locator polynomials meaning $\lambda$ must be to. ■

**Proposition 3.13**

Power decoding satisfies the ML certification property

*Proof.* Let $c$ be a codeword decoded from $r$, where the minimal solution is $(\lambda, \psi^{(1)}, \ldots, \psi^{(l)})$. Due to Proposition 3.12, $\lambda$ must have the form $\lambda = \prod_{i \in I}(x - a_i)$ for some $I \subseteq \{1, \ldots, n\}$. Then

$$d(c, r) = |I| = \deg \lambda.$$

Thus $\deg \lambda$ is minimal among all error locator polynomials if and if $d(c, r)$ where $c$ is a codeword. Since $\lambda$ is always minimal, the distance $d(c, r)$ must also be.

■

So now we know that whenever the Power decoding gives us a codeword, it must be a closest codeword. This is useful because if we can count the errors that makes the received word closer to another codeword then the one send, we can calculate the probability of decoding into wrong codeword. This result is seen in Theorem 3.15 but before proving it we need a lemma. The lemma and the theorem are both based on (Schmidt et al. 2006),

**Lemma 3.14**
Let $S_1$ and $S_2$ be two spheres in the Hamming space $\mathbb{F}_q^n$ with same centre. Let the radius of $S_1$ and $S_2$ be respectively $r_1$ and $r_2$. Finally let $v$ be a point on the surface of $S_1$, and create the sphere $S_3$ with radius $\rho$ and centre $v$. Then amount of the points on the intersection of the surfaces $S_2$ and $S_3$ is given by

$$U(q, r_1, r_2\rho) = \sum_{i=\left\lceil \frac{r_1+r_2-\rho}{2} \right\rceil}^{r_1+r_2-\rho} \binom{r_1}{i} \binom{i}{\rho - (r_1 + r_2) + 2i} \binom{n - r_1}{r_2 - i} (q-2)^{\rho-(r_1+r_2)+2i} (q-1)^{r_2-i}$$

*Proof.* For a vector $u = (u_1, \ldots, u_n)$ we define the support $\sup(u)$ and the support complement $\sup(u)^c$ in the following way

$$\sup(u) = \{i \in \{1, \ldots, n\} \mid u_i \neq 0\}$$
$$\sup(u)^c = \{1, \ldots, n\} \backslash \sup(u) = \{i \in \{1, \ldots, n\} \mid u_i = 0\}$$

Assume without lost of generality the centre of $S_1$ and $S_2$ is 0. Thus $r_1$ is the Hamming weight of $v$, and therefore $v$ consists of $r_1$ nonzero entries. This means that $|\sup(v)| = r_1$ and $|\sup(v)^c| = n - r_1$. Consider a point $w$ on $S_2$, which for the same reason have support $\sup(w) = r_2$. When $v$ is fixed, we divide the the indices of $w$ in the following sets

$$I = \sup(w) \cap \sup(v) = \{j \mid v_j \neq 0 \neq w_j\}$$
$$J = \sup(w) \cap \sup(v)^c = \{j \mid v_j = 0 \neq w_j\}$$
$$\Delta = \sup(w) \cap \sup(v) \cap \sup(v - w) = \{j \mid v_j \neq 0 \neq w_j \neq v_j\}$$

Define $|I| = i$ and $|\Delta| = \delta$. If we want to compute $d(v, w) = |\sup(v-w)|$, we should count the elements of $\sup(v)$ and $\sup(w)$. However we have counted two many since we have counted every element of $I$ twice, and some might not contribute to $d(v, w)$. Thus we subtract $2i$ and add $\delta$ such that we get

$$d(v, w) = r_1 + r_2 - 2i + \delta.$$

The aim of the proof is to find how many $w$ satisfies $d(v, w) = \rho$ for our fixed $v$. Let us also fix $i$, meaning

$$\delta = \rho - (r_1 + r_2) + 2i.$$

We will now count the possibilities of choosing $I$, $J$ and $\delta$ as will as the entries $w$.

Firstly, there is $\binom{i}{\delta}$ ways of choosing $\Delta$ given $I$. For $j \in \Delta$ the entry $w_j$ can any element in $\mathbb{F}_q$ besides $v_j$ and 0. Thus we have a factor of

$$\binom{i}{\delta}(q-2)^{\delta} = \binom{i}{\rho - (r_1 + r_2) + 2i}(q-2)^{\rho-(r_1+r_2)+2i}. \tag{25}$$

Next, there is $\binom{r_1}{i}$ ways of choosing $I$ given $\sup(v)$. Note that for $j \in I \backslash \Delta$ the entry $w_j = v_j$, and for $j \in I \cap \Delta$ we have already determined $w_j$. Thus this give another factor of

$$\binom{r_1}{i} \tag{26}$$

Finally, there is $\binom{n-r_1}{r_2-i}$ ways of choosing $J$ given $\sup(v)^c$. For $j \in J$ the entry $w_j$ must be nonzero. Thus we have the factor

$$\binom{n-r_i}{r_2-i}(q-1)^{r_2-i}. \tag{27}$$

So we only need to sum the product of (25), (26) and (27) for all possible $i$. To give a bound on $i$, we use $0 \leq \delta \leq i$ and hereby $0 \leq \rho - (r_1 + r_2) + 2i \leq i$. This implies that

$$\frac{r_1 + r_2 - \rho}{2} \leq i \quad \text{and} \quad i \leq r_1 + r_2 - \rho$$

Therefore we get that there is

$$\sum_{i=\lceil \frac{r_1+r_2-\rho}{2} \rceil}^{r_1+r_2-\rho} \binom{r_1}{i}\binom{i}{\rho - (r_1 + r_2) + 2i}\binom{n-r_1}{r_2-i}(q-2)^{\rho-(r_1+r_2)+2i}(q-1)^{r_2-i}$$

possibilities of choosing $w$. ■

Before stating Theorem 3.15, we will need a new definition. The weight distribution $A_w(c)$ is the number of codewords which are within distance $w$ from $c$. For linear codes, we have that $A_w(c)$ is the same for all possible codewords $c$, thus we will use the notation $A_w$ for the weight distribution. Clearly $A_0 = 1$ and $A_w = 0$ for $w = 1, \ldots, d-1$. Furthermore for an MDS code such as a Reed Solomon codes, it is given by

$$A_w = \binom{n}{w}\sum_{j=0}^{w-d}(-1)^j\binom{w}{j}(q^{w-d+1-j} - 1)$$

for $w \geq d$. (Proposition 3.2.20 in Pellikaan et al. 2018)

**Theorem 3.15**

Let $C$ be a linear code with length $n$, dimension $k$, and minimum distance $d$ over $\mathbb{F}_q$. Let $c \in C$ and generate an error message $e$ uniformly of weight $\varepsilon$. Assume that we want to correct the errors from the received message $c + e$ using a decoder satisfying the ML certificate. Then the probability $P_e(\varepsilon)$ for a decoding a wrong codeword is bounded by

$$P_e(\varepsilon) \leq \frac{\sum_{w=d}^{\varepsilon+\varepsilon_{\min}} A_w \sum_{\rho=0}^{\max\{\varepsilon,\varepsilon_{\max}\}} U(q,\varepsilon,w,\rho)}{\binom{n}{\varepsilon}(q-1)^{\varepsilon}}, \tag{28}$$

where $\varepsilon_{\max}$ is the decoding radius, $A_w$ is the weight distribution, and $U$ is given as in 3.14.

*Proof.* Consider two codewords $c, c' \in C$ with Hamming distance $d(c, c') = w$. Let $e \in \mathbb{F}_q^n$ be of Hamming weight $\varepsilon$, and consider the recieved message $r = c + e$. We will count for how many such $e$ makes $r$ decoded as $c'$.

Consider the two spheres $s$ and $s'$, where $s$ is centred in $c$ with radius $\varepsilon$, and $s'$ is centred in $c'$ with radius $\varepsilon_{\max}$. Note that $r$ can only be a point on the surface of $s$. If $r$ is decoded as $c'$, $r$ must also be within the sphere $s'$, since it should be possible to decode $r$ as $c'$. However the distance from $r$ and $c'$ must be at most $\varepsilon$, since otherwise $r$ would be decoded as $c$ due to the ML certificate property.

Therefore we shall count the points on $s$ which is within distance $\rho$ to $c'$, where $0 \leq \rho \leq \min\{\varepsilon, \varepsilon_{\max}\}$. To do so we consider two new spheres $s_1$, $s_2$ both centred in $c$, where $s_2$ has radius $w$, and $s_1$ has radius $\varepsilon$, i.e. $s = s_1$. Note that $c'$ is a point on $s_1$. The points we are searching for is those points on $s_2$ which has distance $\rho$ for $0 \leq \rho \leq \min\{\varepsilon, \varepsilon_{\max}\}$. Thus we get by Lemma 3.14 that the number can be computed as

$$N_w = \sum_{\rho=0}^{\min\{\varepsilon,\varepsilon_{\max}\}} U(q,\varepsilon,w,\rho)$$

Thus we we now have a number on how many points might be decoded as $c'$. However we are not interested in just the errors that make $r$ decoded as $c'$ but all codewords which are not $c$. Note that $N_w$ only depend on the amount of errors $\varepsilon$ and the distance $w = d(c, c')$ for a fixed code. So let us fix $\varepsilon$, and try to give a bound on $w$. Due to the definition of minimal distance, we have $d \leq w$. Also if $w$ is greater than $\varepsilon + \varepsilon_{\max}$, the message $r$ cannot be decoded as $c'$.

For a given $w$ there is at $A_w$ codewords with distance $w$ to $c'$. Thus we get that there is at most

$$\sum_{w=d}^{\varepsilon+\varepsilon_{\max}} A_w N_w \tag{29}$$

Note there might be some points we have counted multiple times, but since we are trying to give an upper bound, this is not a problem.

If we want to calculate the probability of decoding error, we have the favourable outcomes bounded by (29). The number of possible outcomes is given as the number of points on the Hamming sphere of radius $\varepsilon$. This must be $\binom{n}{\varepsilon}(q-1)^{\varepsilon}$ since we have to choose $\varepsilon$ nonzero entries out of $n$. This implies (28).

■

# 4 Multi-sequence linear feedback shift-register synthesis

In power decoding it is required to solve the powered key equation. We will now look into how to this based on a method of solving MgLFSR problems presented in (Nielsen 2013a).

Consider $l$ polynomials $S_1, \dots S_l \in \mathbb{F}[x]$ over some field $\mathbb{F}$, and let $m_1, \dots, m_l \in \mathbb{N}_+$. The multi-sequence linear feedback shift-register synthesis problem (MLFSR) is the problem of finding a monic polynomial $\Lambda \in \mathbb{F}[x]$ of lowest possible degree such that there exist $\Omega_1, \dots \Omega_l \in \mathbb{F}[x]$ satisfying

$$\begin{aligned} &\Lambda S_i \equiv \Omega_i \mod x^{m_i} \\ &\deg \Lambda > \deg \Omega_i \end{aligned} \quad \text{for } i = 1 \dots l$$

For our usage, we are not interested in this problem, and instead we will look at a generalised version. Consider $S_1, \dots, S_l, G_1, \dots, G_l \in \mathbb{F}[x]$, let $\nu \in \mathbb{N}_+$ and $w_0 \dots w_1 \in \mathbb{N}_0$. The generalised multi-sequence linear feedback shift-register synthesis problem (MgLFSR) is the problem of finding a monic polynomial $\Lambda \in \mathbb{F}[x]$ of lowest possible degree such that there exist $\Omega_1, \dots \Omega_l \in \mathbb{F}[x]$ satisfying

$$\begin{aligned} &\Lambda S_i \equiv \Omega_i \mod G_i \\ &\nu \deg \Lambda + w_0 > \nu \deg \Omega_i + w_i \end{aligned} \quad \text{for } i = 1 \dots l \tag{30}$$

We will call $(\Lambda, \Omega_1, \dots, \Omega_l) \in \mathbb{F}[x]^{l+1}$ a solution if it satisfies (30). In particular, if $\Lambda$ is monic and of lowest degree, i.e. $\deg \Lambda \leq \deg \Lambda'$ for any other solution $(\Lambda', \Omega'_1, \dots, \Omega'_l)$, we will call it minimal solution.

Furthermore we will assume $\deg S_i \leq \deg G_i$, since otherwise one can substitute $S_i$ by $(S_i \mod G_i)$, and it will yield the same solutions. Also we will assume $w_0 < \max_i\{\deg S_i + w_i\}$, since otherwise we will have the minimal solution $(1, S_1, \dots S_l)$.

Since we will deal with matrices over $\mathbb{F}[x]$, we will explain some of the notation

- We will often consider a matrix $V$ in $\mathbb{F}[x]^{(l+1)\times(l+1)}$. We will refer to the rows as $v_0, \dots, v_l$, and for each $i = 0, \dots, l$, we have $v_i = [v_{i0}, \dots, v_{il}]$, i.e.

$$V = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_l \end{bmatrix} = \begin{bmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,l} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,l} \\ \vdots & \vdots & \ddots & \vdots \\ v_{l,0} & v_{l,1} & \cdots & v_{l,l} \end{bmatrix}.$$

  where each $v_{ij} \in \mathbb{F}[x]$.

- For a polynomial $v_{ij} \in \mathbb{F}[x]$ the degree is defined in the usual way. For a vector $v_i = [v_{i0}, \dots, v_{il}]$ we define the degree as the maximum of the polynomial in it, i.e. $\deg(v_i) =$

$\max\{\deg v_{ij} \mid j = 0, \ldots, l\}$. For a matrix $V$ we define it as the sum of the degree of the rows, i.e. $\deg(V) = \sum_{i=0}^{l} \deg v_i$.

- The leading position of a vector $v_i$ is $LP(v_i) = \max\{j \mid \deg v_j = \deg v\}$. The leading term of $v_i$ is $LT(v) = v_{i,LP(v_i)}$. Note that leading term gives a polynomial and not just a term.

**Example 4.1.** Consider the matrix

$$V = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} x^3 & 1 & x \\ x & x^2+1 & 0 \\ 1 & x & x+1 \end{bmatrix}.$$

Using notation of this section we have that $\deg(v_0) = \max\{\deg x^3, \deg 1, \deg x\} = 3$, $\deg(v_1) = 2$, $\deg(v_2) = 1$. Thus $\deg(V) = 3 + 2 + 1 = 6$. To find the leading position we have to look at where a polynomial of maximal degree occurs. Doing so, we see $LP(v_0) = 0$ since the polynomial in $0^{\text{th}}$ entry has degree greater than any other in that row. Similarly $LP(v_1) = 2$. For $LP(v_2)$ we see that two polynomials of degree 1 in the row, and thus we have to take the highest position, i.e. $LP(2) = 2$. Finally we have $LT(v_0) = x^3$, $LT(v_1) = x^2 + 1$, $LT(v_2) = x + 1$.

$\mathfrak{D}$

**Definition 4.2**
Let $(R, +, \cdot)$ be a ring with multiplicative identity $1_R$, and let $(M, +)$ be an Abelian group. Consider an operator $\cdot : R \times M \to M$. The group $M$ is a module over $R$ if for all $r, s \in R$ and $m, n \in M$

- $r \cdot (m + n) = r \cdot m + r \cdot n$
- $(r \cdot s) \cdot m = r \cdot (s \cdot m)$
- $(r + s) \cdot m = r \cdot m + s \cdot m$
- $1_R \cdot m = m$

**Remark 4.3.** We use notation $+$ for two operations but keep in mind that they may be different. The same can be said for $\cdot$.

A module is free if there exist a set $V = \{v_1, \ldots, \}$ which is generating set, i.e. that every element $m \in M$ can be written as $m = \alpha_1 v_1 + \cdots + \alpha_n v_n$ for some $n \in \mathbb{N}$ and $\alpha_1, \ldots, \alpha_n \in R$.

If $V$ is linearly independent, i.e. $0 = \alpha_1 v_1 + \dots \alpha_n v_n$ implies $\alpha_1 = \cdots = \alpha_n = 0$, we will call $V$ a basis. In case where the basis is finite we often order $V$ in a vector $V = [v_1, \dots, v_n]^T$. Then in cases where the basis consist of vectors or tuples, the basis $V$ will be a matrix whose rows are linearly independent and a generating set.

With these definitions in mind, we will now go back to the problem of solving MgLFSR. Here we will consider the set

$$M = \{(\Lambda, \Omega_1, \dots \Omega_l) \in \mathbb{F}[x] \mid \Lambda S_i \equiv \Omega_i \mod G_i \text{ for } i = 1, \dots, l\}$$

Notice $M$ is a module over $\mathbb{F}[x]$ where the underlying operations are given naturally as if $M$ was a vector space. Also the solutions to (30) are elements of $M$. In fact the solutions are the elements of $M$ that satisfies $\nu \deg \Lambda + w_0 > \nu \deg \Omega_i + w_i$. Thus our goal is now to determine whether an element of $M$ satisfies these degree constrictions. For this we will first need a basis of $M$.

**Proposition 4.4**

The module $M$ is a free module as has the following basis

$$V = \begin{bmatrix} 1 & S_1 & S_2 & & S_l \\ 0 & G_1 & 0 & \cdots & 0 \\ 0 & 0 & G_2 & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & G_l \end{bmatrix} \tag{31}$$

*Proof.* Clearly each row of $V$ is in $M$. So let $m = (m_0, \dots, m_l) \in M$. Then $m_0 S_i \equiv m_i \mod G_i$ for $i = 1, \dots, l$, meaning there exists $p_1, \dots, p_l \in \mathbb{F}[x]$ such that $m_i = m_0 S_i + p_i G_i$ for $i = 1, \dots, l$. This means

$$m = m_0 v_0 + p_1 v_1 + \cdots p_l v_l$$

where $v_j$ is the $j^{\text{th}}$ row of $V$. Thus every element of $M$ can be written as linear combinations of rows in $V$. Finally the basis is linearly independent since it is a triangular matrix. ■

To combat the difficulty of solving MgLFSR gives us compared to solving MLFSR we will introduce the map $\Phi : \mathbb{F}[x]^{l+1} \to \mathbb{F}[x]^{l+1}$ by

$$\Phi : (p_0, p_1, \dots p_l) \mapsto (x^{w_0} p_0(x^\nu), x^{w_1} p_1(x^\nu), \dots x^{w_l} p_l(x^\nu)) \tag{32}$$

Notice that in case of MLFSR, $\Phi$ is the identity map. For a matrix $(v_0, \dots, v_l)^T$ with rows in $\mathbb{F}[x]^{l+1}$, we extend the map entrywise as

$$\Phi((v_0, v_1, \dots, v_l)^T) = (\Phi(v_0), \Phi(v_1), \dots, \Phi(v_l))^T$$

Later we will use the inverse of the map $\Phi^{-1}$, which is given by

$$\Phi^{-1} : (q_0, q_1, \ldots, q_l) \mapsto (x^{w_0/\nu} q_0(x^{1/\nu}), x^{w_1/\nu} q_1(x^{1/\nu}), \ldots x^{w_l/\nu} q_l(x^{1/\nu}))$$

The set $\Phi(M) = \{\varphi(m) \mid m \in M\}$ is still a module but now over $\mathbb{F}[x^\nu]$ instead of $\mathbb{F}[x]$. If $V$ is a basis of $M$ then $\Phi(V)$ is a a basis of $\Phi(M)$. Likewise if $W$ is a basis of $\Phi(M)$ then $\Phi^{-1}(W)$ is a basis of $M$. Thus working with $M$ or $\Phi(M)$ is almost identical. The usefulness of choosing $\Phi(M)$ instead of $M$ can be seen in the following lemma.

**Lemma 4.5**
An element $s \in M$ is a solution to MgLFSR if and only if

- $LP(\Phi(s)) = 0$

Furthermore a non-zero $s \in M$ is a minimal solution to MgLFSR if and only if

- $LP(\Phi(s)) = 0$, and
- for all non-zero $b \in M$ with $LP(\Phi(b)) = 0$ it holds that $\deg \Phi(s) \leq \Phi(b)$

*Proof.* Let $s = (\Lambda, \Omega_1, \ldots, \Omega_l) \in M$. Then we have the following equivalences

$$\begin{aligned}
& s \text{ is a solution} \\
\Leftrightarrow \quad & \nu \deg \Lambda + w_0 > \nu \deg \Omega_i + w_i \qquad \text{for } i = 1, \ldots l \\
\Leftrightarrow \quad & \deg(x^{w_0}\Lambda(x^\nu)) > \deg(x^{w_i}\Lambda(x^\nu)) \quad \text{for } i = 1, \ldots l \\
\Leftrightarrow \quad & LP(\Phi(s)) = 0.
\end{aligned}$$

This proves the first part.

Let $b = (b_0, \ldots, b_l) \in M$ such that $LP(\Phi(b)) = 0$. Then juastas previously, we have $\nu \deg b_0 + w_0 > \nu \deg b_i + w_i$ for $i = 1, \ldots, l$. Then

$$\deg(\Phi(b)) = \max_{i=0,\ldots,l} \{\deg(x^{w_i} b(x^\nu))\} = \max_{i=0,\ldots,l} \{\nu \deg b_i + w_i\} = \nu \deg b_0 + w_0.$$

Therefore $\deg(\Lambda) \leq \deg(b_0)$ if and only if $\deg(\Phi(s)) \leq \deg(\Phi(b))$, where $s = (\Lambda, \Omega_1, \ldots, \Omega_l)$. This implies the lemma. ■

**Definition 4.6**
A matrix $V \in \mathbb{F}[x]^{(l+1)\times(l+1)}$ is in weak Popov form if

- its row are linearly independent, and
- the leading positions of the rows are pairwise different.

**Lemma 4.7**

Let $V \in \mathbb{F}[x]^{(l+1)\times(l+1)}$ be a basis over a module $M$ in weak Popov form, and let $b \in M$. Then $\deg v \leq \deg b$ for the $v$ in $V$ such that $LP(v) = LP(b)$

*Proof.* Since $V$ is a basis of $M$ there must exist $p_0, \ldots, p_l \in \mathbb{F}[x]$ such that $b = p_0 v_0 + \cdots + p_l v_l$. For $i$ such that $p_i \neq 0$, we have that

$$LP(p_i v_i) = \max\{j \mid \deg(p_i v_{i,j}) = \deg(p_i v_i)\} = \max\{j \mid \deg(v_{i,j}) = \deg(v_i\} = LP(v_i).$$

Define $(i_0, j_0)$ as $LP(b) = j_0$ and $LP(v_{i_0}) = j_0$. Note that $(i_0, j_0)$ is defined uniquely. Assume for contradiction $p_{i_0} = 0$. Since $LP(b) = j_0$ and $b \neq 0$ there must exist $i_1$ such that $p_{i_1} v_{i_1,j_0} \neq 0$; in fact we choose $i_1$ such that it maximises the degree of $p_{i_1} v_{i_1,j_0}$. Define $j_1$ as $LP(v_{i_1}) = j_1$, and note that $j_0 \neq j_1$ since $V$ is in weak Popov form. However this will imply $LP(p_{i_1} v_{i_1}) = j_1$. Therefore there most exist $i_2$ such that $p_{i_2} v_{i_2 j_1}$ cancel out the highest term of $p_{i_1} v_{i_1 j_1}$. Define $j_2 = LP(v_{i_2})$. Continue this process to find new $(i_k, j_k)$. This must eventually terminate since the basis is finite. At that point we have found a leading position different found $j_0$, which highest terms cannot be cancelled. This contradicts with $LP(b) = j_0$, and thus $p_{i_0} \neq 0$.

Furthermore one can also proof that the leading term of $p_{i_0} v_{i_0,j_0}$ cannot be cancelled out with some other term using similar techniques as proving $p_{i_0} \neq 0$. Thus we have

$$\deg b = \deg p_{i_0} v_{i_0,j_0} = \deg p_{i_0} + \deg v_{i_0} \geq \deg v_{i_0}$$

■

**Proposition 4.8**

Let $\Phi(V)$ be on weak Popov form, and let $v$ be the row in $\Phi(V)$ such that $LP(v) = 0$. Then $\Phi^{-1}(v)$ is a minimal solution.

*Proof.* We will now combine Lemma 4.5 and 4.7. Assume $\Phi(V)$ is on weak Popov form, and consider the row of $v$ in $\Phi(V)$ such that $LP(v) = 0$. Then we know from Lemma 4.7 that $\deg v \leq \deg b$ for all nonzero $b$ such that $LP(b) = LP(v)$. This combined with Lemma 4.5 shows that $\Phi^{-1}(v)$ is a minimal solution. ■

Thus in light of Proposition 4.8, our goal is now to find a way to turn a matrix into one in Popov form.

A row reduction on $V$ is defined in the following way. Find two rows $v_i, v_j$ such that $\deg v_i \leq \deg v_j$ and $LP(v_i) = LP(v_j)$. Then replace $v_j$ by $v_j - \alpha x^\delta v_i$, where $\alpha \in \mathbb{F}$ and $\delta \in \mathbb{N}$ are chosen such that the leading terms are cancelled out.

We will use row reductions to compute a basis in Popov form. Here we will start of with a matrix and apply row reductions on it until it is no longer possible. However for this the process most eventually terminate and row reduced basis must span the same module as the original. These problems are covered in Lemma 4.9 and Lemma 4.10.

**Lemma 4.9**
Define the value function $\varphi : \mathbb{F}[x]^{l+1} \to \mathbb{N}$ given by

$$\varphi : v \mapsto (l+1)\deg v + LP(v).$$

If $v_j$ is replaced by $v'_j$ under a row reduction then $\varphi(v'_j) < \varphi(v_i)$

*Proof.* Note that $\deg v'_j \leq \deg v_j$ since $v'_j = v_j - \alpha x^\delta v_i$ and $LP(v_i) = LP(v_j)$. If $\deg v'_j < \deg v_j$ we have that

$$\begin{aligned}
\varphi(v'_j) &= (l+1)\deg v'_j + LP(v'_j) \\
&\leq (l+1)(\deg v_j - 1) + LP(v'_j) \\
&= (l+1)\deg v_j + LP(v'_j) - (l+1) \\
&< (l+1)\deg v_j \\
&\leq (l+1)\deg v_j + LP(v_j) = \varphi(v_j)
\end{aligned}$$

On the other hand, if $\deg v'_j = \deg v_j$. Define $h = LP(v_j) = LP(v_i)$. Due to the definition of leading position, we have

$$\begin{aligned}
\deg v_{jk} &\leq \deg v_{jh} \quad \text{for } k = 0, \ldots, h \\
\deg v_{jk} &< \deg v_{jh} \quad \text{for } k = h+1, \ldots, l
\end{aligned}$$

Similar inequalities can be shown for $v_i$, implying

$$\begin{aligned}
\deg x^\delta v_{ik} \leq \deg x^\delta v_{ih} = \deg v_{jh} \quad \text{for } k = 0, \ldots, h \\
\deg x^\delta v_{ik} < \deg x^\delta v_{ih} = \deg v_{jh} \quad \text{for } k = h+1, \ldots, l.
\end{aligned}$$

Thus when we compute the entries of $v'_j = v_j - \alpha x^\delta v_i$, the $(l-k)^{\text{th}}$ last entries cannot have degree $\deg(v_j)$. The $k^{\text{th}}$ cannot have degree $\deg(v_j)$ due to the choice of $\alpha x^\delta$. Since $\deg(v'_j) = \deg(v_j)$, there must exist an entry with lower index than $k$ that has degree $\deg(v_j)$. Therefore $LP(v'_j) < LP(v_j)$. ■

**Lemma 4.10**
Let $V \in \mathbb{F}[x]^{(l+1)\times(l+1)}$ be a basis of a module $M$. Apply one row reduction to $V$ and get $V'$. Then $V'$ is a basis of $M$.

*Proof.* Let $b \in M$, and write this as $b = p_0 v_0 + \cdots + p_l v_l$. Assume without loss of generality that $i = 0$ and $j = 1$. Then the only change to the basis is $v_1' = v_1 - \alpha x^\delta v_0$. Then we can write $b$ as

$$\begin{aligned} b &= p_0 v_0 + p_1 v_1 + p_2 v_2 + \cdots + p_l v_l \\ &= p_0 v_0 + p_1 (v_1' + \alpha x^\delta v_0) + p_2 v_2 + \cdots + p_l v_l \\ &= (p_0 + p_1 \alpha x^\delta) v_0 + p_1 v_1' + p_2 v_2 + \cdots + p_l v_l \end{aligned}$$

Thus $v_0, v_1', v_2, \ldots v_l$ is also a basis. ■

In light of Lemma 4.9 and 4.10, we know that the process of row reducing a basis must eventually terminate, and the reduced basis spans the same module as the original. Notice we can make a row reduction when there exist $i \neq j$ such that $LP(v_i) = LP(v_j)$. Thus when we no longer can do row reductions the rows must have different leading positions. Thus it is on Popov form.

Our strategy for solving MgLFSR is to write up a basis, modify the problem with $\Phi$, apply row reductions, and finally extract the row with 0 as the leading position. This is formalise in the following algorithm.

**Algorithm 3** Mulder-Storjohann algorithm

**Input:** Polynomials $S_1, \ldots S_l, G_1, \ldots, G_l \in \mathbb{F}[x]$ and weights $\nu \in \mathbb{N}_+, w_0, \ldots, w_l \in \mathbb{N}_0$

**Output:** The minimal solution $(\Lambda, \Omega_1, \ldots, \Omega_l)$ to MgLFSR given in (30).

Define $M$ as in (31)

Compute $\Phi(M)$ where $\Phi$ is given in (32)

Apply row reduction on $\Phi(M)$ until it is no longer possible

Find the the row $v$ in the reduced $\Phi(M)$ with $LP(v) = 0$

**return** $\Phi^{-1}(v)$

**Proposition 4.11**

Algorithm 3 eventually terminates and it is correct.

*Proof.* The algorithm terminates due to Lemma 4.9. More precisely in each iteration of row reduction $\varphi$-value of the matrix will decrease due to lemma. It follows from Proposition 4.8 that $\Phi^{-1}(v)$ is in fact a minimal solution. ■

## 4.1 Application to Power decoding

Recall that in Power decoding we need to solve the problem of

$$\begin{aligned} \lambda R^{(t)} &\equiv \psi^{(t)} \mod G \\ \deg \psi^{(t)} &\leq \lambda + t(k-1) \end{aligned} \quad \text{for } t = 1, \ldots, l. \tag{33}$$

We will now explain how to solve this using the techniques developed in this section. So firstly we will outline why (33) is an special instance of (30). Obviously $i = t$, $\Lambda = \lambda$, $S_i = R^{(t)}$, $\Omega_i = \psi^{(t)}$ and $G_i = G$ is constant. Regarding the degree constriction, we use $\nu = 1$, $w_0 = l(k-1)+1$ and $w_i = (l-t)(k-1)$ since

$$\begin{aligned} \deg \psi^{(t)} &\leq \deg \lambda + t(k-1) \\ \deg \psi^{(t)} - t(k-1) &\leq \deg \lambda \\ \deg \psi^{(t)} + (l-t)(k-1) &< \deg \lambda + l(k-1) + 1. \end{aligned}$$

The first step of algorithm 3 is to construct the matrix

$$M = \begin{bmatrix} 1 & R^{(1)} & R^{(2)} & & R^{(l)} \\ 0 & G & 0 & \cdots & 0 \\ 0 & 0 & G & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & G \end{bmatrix}$$

and apply the $\Phi$ map to it such that we ovtain

$$\Phi(M) = \begin{bmatrix} x^{l(k-1)+1} & x^{(l-1)(k-1)}R^{(1)} & x^{(l-2)(k-1)}R^{(2)} & & R^{(l)} \\ 0 & x^{(l-1)(k-1)}G & 0 & \cdots & 0 \\ 0 & 0 & x^{(l-2)(k-1)}G & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & G \end{bmatrix} \tag{34}$$

Now one should apply row reductions on this matrix until it is no longer possible. The row with leading position 0, should induce a solution to (33). How this is done in practice can be seen in the following example.

**Example 4.12.** Recall Example 3.3 where we had $q = 7$, $n = 5$, $k = 2$, and $l = 2$. At some point we wanted to solve

$$\begin{aligned} \lambda R^{(t)} &\equiv \psi^{(t)} \mod G \\ \deg \psi^{(t)} &\leq \deg \lambda - t \end{aligned} \quad \text{for } t = 1, 2 \tag{35}$$

where $\lambda$ and $\psi^{(t)}$ is unknown and

$$\begin{aligned} G &= x^5 + 6x^4 + x^3 + 6x^2 + x + 6 \\ R^{(1)} &= 3x^4 + 4x^3 + 2x^2 + 3 \\ R^{(2)} &= 3x^4 + x^3 + 2x^2 + x + 4 \end{aligned} \tag{36}$$

As in Algorithm 3, we first compute $M$ and $\Phi(M)$. By directly use (34), we obtain

$$\begin{aligned} \Phi(M) &= \begin{bmatrix} x^3 & xR^{(1)} & R^{(2)} \\ 0 & xG & 0 \\ 0 & 0 & G \end{bmatrix} \\ &= \begin{bmatrix} x^3 & 3x^5 + 4x^4 + 2x^3 + 3x & 3x^4 + x^3 + 2x^2 + x + 4 \\ 0 & x^6 + 6x^5 + x^4 + 6x^3 + x^2 + 6x & 0 \\ 0 & 0 & x^5 + 6x^4 + x^3 + 6x^2 + x + 6 \end{bmatrix} \end{aligned}$$

We will now apply row reductions to this matrix until it is no longer possible. In first iteration we see that both $0^{\text{th}}$ and $1^{\text{st}}$ row have leading position 1. Therefore we should multiply the $0^{\text{th}}$ row by $-3^{-1} = 2$ and add to row 1. Contuing this process we get

$$\begin{aligned} \Phi(M) &= \begin{bmatrix} x^3 & 3x^5 + 4x^4 + 2x^3 + 3x & 3x^4 + x^3 + 2x^2 + x + 4 \\ 0 & x^6 + 6x^5 + x^4 + 6x^3 + x^2 + 6x & 0 \\ 0 & 0 & x^5 + 6x^4 + x^3 + 6x^2 + x + 6 \end{bmatrix} \\ &\sim \begin{bmatrix} x^3 & 3x^5 + 4x^4 + 2x^3 + 3x & 3x^4 + x^3 + 2x^2 + x + 4 \\ 2x^4 & 5x^4 + 6x^3 + 6x & 6x^5 + 2x^4 + 4x^3 + 2x^2 + x \\ 0 & 0 & x^5 + 6x^4 + x^3 + 6x^2 + x + 6 \end{bmatrix} \\ &\sim \begin{bmatrix} x^3 & 3x^5 + 4x^4 + 2x^3 + 3x & 3x^4 + x^3 + 2x^2 + x + 4 \\ 2x^4 & 5x^4 + 6x^3 + 6x & x^4 + 5x^3 + x^2 + 2x + 6 \\ 0 & 0 & x^5 + 6x^4 + x^3 + 6x^2 + x + 6 \end{bmatrix} \\ &\sim \begin{bmatrix} x^3 & 3x^5 + 4x^4 + 2x^3 + 3x & 3x^4 + x^3 + 2x^2 + x + 4 \\ 2x^4 & 5x^4 + 6x^3 + 6x & x^4 + 5x^3 + x^2 + 2x + 6 \\ 5x^5 & 2x^5 + x^4 + x^2 & x^4 + 4x^2 + 2x + 6 \end{bmatrix} \\ &\sim \begin{bmatrix} 3x^5 + x^3 & 6x^4 + 2x^3 + 2x^2 + 3x & 5x^4 + x^3 + 3x^2 + 5x + 2 \\ 2x^4 & 5x^4 + 6x^3 + 6x & x^4 + 5x^3 + x^2 + 2x + 6 \\ 5x^5 & 2x^5 + x^4 + x^2 & x^4 + 4x^2 + 2x + 6 \end{bmatrix} \end{aligned}$$

In the matrix we end up with, we see that the leading positions of row, 0, 1 and 2 are

respectively 0, 2 and 1. Therefore we should apply $\Phi^{-1}$ to the 0th. Hence we get the solution

$$\begin{aligned}&\Phi^{-1}\big(3x^5+x^3,\ 6x^4+2x^3+2x^2+3x,\ 5x^4+x^3+3x^2+5x+2)\big)\\ =&(3x^2+1,\ 6x^3+2x^2+2x+3,\ 5x^4+x^3+3x^2+5x+2)\end{aligned}$$

Remember that that a minimal solution need a monic $\lambda$. Thus the only calculation left is to divide each entry by 3 such that we get the minimal solution

$$\begin{aligned}\lambda &= x^2+5\\ \psi^{(1)} &= 2x^3+3x^2+3x+1\\ \psi^{(2)} &= 4x^4+5x^3+x^2+4x+3\end{aligned}$$

# 5 Simulations

In this section we will deal with Reed Solomon codes $RS(q, n, k)$ and try to estimate the various probabilities of Power Decoding when choosing different $l$ and $\varepsilon$. We will do so by simulations of $N$ experiments, where in each experiment we will generate an error message $e$ of weight $\varepsilon$. We will choose the message $m = 0$ and for each experiment we will add $e$ such that we receive $r = m + e$ and decode $r$ as $\hat{m}$. Remember that there is 3 outcomes: $\hat{m}$ can either be $m$, $\hat{m}$ can be a message different from $m$ or the method declares failure. $N_s$ is the number experiments where the method success, ie. $\hat{m} = m$. $N_f$ is the number of experiments where the method fails, i.e. $\hat{m} \neq m$ or the method declares failure. The number $N_e$ is the number of decoding error, i.e. $\hat{m} \neq m$. Note that $N = N_s + N_f$ and $N_e \leq N_f$. From this we can estimate the probability of success by the frequency of success $P_s = N_s/N$. Similar we will use the frequency of failure $P_f = N_f/N$ and the frequency of decoding error $P_e = N_e/N$. These computations sage, and the code for these can be seen in Appendix A.

The first simulation we will show is on $RS(31, 16, 2)$, meaning that the rate is $k/n = 1/8$. For a given $l$ and $\varepsilon$ we will calculate frequency of success, i.e. $m = \hat{m}$. The values of $\varepsilon$ will run from $d/2 = 8$ up to 11, and $l$ will be all possible, namely $l = 2, \ldots, 15$. For each pair $(\varepsilon, l)$ we will do $N = 10^3$ experiments. The results of these simulations can be seen in Table 3. The simulation seems to indicate that whenever $l$ gets bigger we are able to correct more errors.

| | | $\varepsilon$ | | | |
|---|---|---|---|---|---|
| | | 8 | 9 | 10 | 11 |
| l | 2 | 1 | 0.959 | 0 | 0 |
| | 3 | 1 | 1 | 0.001 | 0 |
| | 4 | 1 | 1 | 0.954 | 0 |
| | 5 | 1 | 1 | 0.965 | 0 |
| | 6 | 1 | 1 | 0.97 | 0 |
| | 7 | 1 | 1 | 0.972 | 0 |
| | 8 | 1 | 1 | 0.95 | 0 |
| | 9 | 1 | 0.999 | 0.96 | 0 |
| | 10 | 1 | 1 | 0.961 | 0 |
| | 11 | 1 | 1 | 0.959 | 0 |
| | 12 | 1 | 1 | 0.952 | 0 |
| | 13 | 1 | 1 | 0.956 | 0 |
| | 14 | 1 | 1 | 0.965 | 0 |
| | 15 | 1 | 1 | 0.965 | 0 |

Table 3: Frequency of success $P_s$ of Power Decoding for $RS(31, 16, 2)$ given $\varepsilon$ and $l$

The code in Table 3 have a somehow small rate. If we instead choose the code $RS(31, 16, 3)$ and $N = 10^4$ we will have a bit higher rate. If we do similar simulation we will end up with the results in Table 4 . Here it seems that it is not possible to correct more errors when $l$ is bigger. However $l = 3$ seems to be better than $l = 2$ since the frequency of success is a bit higher.

| | | $\varepsilon$ | | |
|---|---|---|---|---|
| | | 7 | 8 | 9 |
| l | 2 | 1 | 0.9665 | 0 |
| | 3 | 1 | 0.9979 | 0 |
| | 4 | 1 | 0.9986 | 0 |
| | 5 | 1 | 0.998 | 0 |
| | 6 | 1 | 0.9983 | 0 |
| | 7 | 1 | 0.9983 | 0 |

Table 4: Frequency of success $P_s$ of Power Decoding for $RS(31, 16, 3)$ given $\varepsilon$ and $l$

In practice, one should also take time complexity into account. When $l$ gets bigger the powered key equation gets more constrictions and therefore harder to solve. For this reason, we will do simulation on various codes and values of $l$. In Table 5, one can see that the average execution time in seconds for each $l$ and code with $N = 10^4$. The most important thing to note is that the average of execution time gets bigger when $l$ gets bigger. Also as one would expect $q = 32$ is faster than 128 since operations on $\mathbb{F}_{32}$ is faster than operations on $\mathbb{F}_{128}$. Finally, it seems that the higher rate codes $RS(32, 16, 6)$ and $RS(32, 4, 2)$ is faster compared to the lower rate code $RS(32, 16, 2)$.

| | | $RS(32,16,2)$ | $RS(128,16,2)$ | $RS(32,16,6)$ | $RS(32,4,2)$ | $RS(31,16,2)$ |
|---|---|---|---|---|---|---|
| | 1 | 0.0093 | 0.0110 | 0.0077 | 0.0026 | 0.0024 |
| | 2 | 0.0200 | 0.0242 | 0.01483 | 0.0047 | 0.004 |
| | 3 | 0.0307 | 0.0417 | 0.01914 | 0.0059 | 0.0063 |
| | 4 | 0.0464 | 0.0683 | | | 0.0105 |
| | 5 | 0.0628 | 0.0844 | | | 0.0117 |
| | 6 | 0.0862 | 0.1003 | | | 0.0159 |
| | 7 | 0.1096 | 0.1323 | | | 0.0167 |
| l | 8 | 0.1102 | 0.1323 | | | 0.0199 |
| | 9 | 0.1265 | 0.1653 | | | 0.02413 |
| | 10 | 0.1519 | 0.2030 | | | 0.0263 |
| | 11 | 0.1615 | 0.2387 | | | 0.0304 |
| | 12 | 0.1770 | 0.2452 | | | 0.0345 |
| | 13 | 0.1890 | 0.2720 | | | 0.0378 |
| | 14 | 0.2091 | 0.2947 | | | 0.0412 |
| | 15 | 0.2252 | 0.3172 | | | 0.0443 |

Table 5: Average execution time in seconds for various codes and $l$. Computations for $l \geq 4$ and one of the codes $RS(32,16,6)$ or $RS(32,16,6)$ are not possible due to the condition $l(k-1) < n$.

In the next time comparison, we will keep the rate fix to $1/4$. We use the codes $(131, 2^{i+2}, 2^i)$ for $i = 1, \ldots, 4$ and $N = 10^4$. The results of these can be seen in table 6. Here we see that the execution time grows when $n$ grows.

| | | l | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 8 | 0.0013 | 0.0024 | 0.0026 | 0.0034 | 0.0041 | 0.0050 | 0.0054 |
| | 16 | 0.0021 | 0.0036 | 0.0047 | 0.0062 | 0.0071 | 0.0074 | 0.0081 |
| n | 32 | 0.0034 | 0.0063 | 0.0085 | 0.0100 | 0.0120 | 0.0123 | 0.0136 |
| | 64 | 0.0059 | 0.0118 | 0.0156 | 0.0205 | 0.0236 | 0.0271 | 0.0317 |
| | 128 | 0.0149 | 0.0281 | 0.0398 | 0.0476 | 0.0591 | 0.0662 | 0.0774 |

Table 6: Average execution time in seconds for the codes $RS(131, n, k)$ with rate $k/n = 1/4$.

# Bibliography

Gao, S. (2002). A new algorithm for decoding reed-solomon codes.

Nielsen, J. S. R. (2013a). Generalised multi-sequence shift-register synthesis using module minimisation. *CoRR*, abs/1301.6529.

Nielsen, J. S. R. (2013b). Power decoding of reed-solomon codes revisited. *CoRR*, abs/1311.1940.

Pellikaan, R., Wu, X.-W., Bulygin, S., and Jurrius, R. (2018). *Codes, Cryptology and CUrves with Computer Algebra*. Cambridge University Press.

Schmidt, G., Sidorenko, V., and Bossert, M. (2006). Collaborative decoding of interleaved reed-solomon codes and concatenated code designs. *CoRR*, abs/cs/0610074.

Schmidt, G., Sidorenko, V., and Bossert, M. (2007). Syndrome decoding of reed-solomon codes beyond half the minimum distance based on shift-register synthesis. *CoRR*, abs/cs/0702130.

von zur Gathen, J. and Gerhard, J. (2013). *Modern Computer Algebra*. Cambridge University Press, third edition.

# A Code for simulations

Firstly we have an example of a setup of the code. Here we look at the Reed-Solomon code $RS(32, 31, 6)$, and the we computes some of the most common objects.

```
sage: q=32
sage: n=31
sage: k=6
sage:
sage: lmax=floor((n-1)/(k-1))
sage: iterations=100
sage:
sage: if (q.is_prime()):
sage:     F=GF(q)
sage:     alpha=vector(range(1,n+1))
sage: elif (q.is_prime_power()):
sage:     F.<irr>=GF(q)
sage:     alpha=vector(irr^i for i in range(0,n))
sage: PolyR.<x> = F[]
sage: d=n-k+1
sage: epsilonmin=floor((d-1)/2)
sage: G=prod([x-a for a in alpha])
```

Now we will move on to some of the used functions we use while simulating. The first vectopol takes a vector $m = (a_1, \ldots, k)$ and turns it into a polynomial $f = a_1 x^k + \cdots + a_l)$. Note that the first entries of $m$ becomes the highest terms of $f$. the function poltovec turns a polynomial $f$ into a vector $f$. Lastly the function encode will encode $m \in \mathbb{F}_q^k$ as $c \in \mathbb{F}_q^k$ where $c = (f(\alpha_1), \ldots, f(\alpha_k))$.

```
sage: def vectopol(m):
sage:     khat=len(m)
sage:     f=0
sage:     for i in range(0,khat):
sage:         f=f+m[i]*x^(khat-i-1)
sage:     return(f)
sage:
sage: def poltovec(f,k):
sage:     m=vector(F,[0 for i in range(0,k)])
sage:     for i in range(0,k):
sage:         (que,rue)=(f).quo_rem(x^(k-i-1))
sage:         m[i]=que
sage:         f=rue
sage:     return(m)
sage:
sage: def encode(m):
sage:     f=PolyR(vectopol(m))
sage:     if f.degree()>=k:
sage:         return("degree of f i too big")
sage:     else:
sage:         c=vector(F,[f(a) for a in alpha])
sage:         return(c)
```

Next is the function makeerror. Giving the input epsilon, it makes a vector $e \in \mathbb{F}_q^n$ of hamming weight epsilon.

```
sage: def makeerror(epsilon):
sage:     e=vector(F,[0 for j in range(0,n)])
sage:     I=list(sample(range(0,n),epsilon))
sage:     for i in I:
sage:         ei=F(0)
sage:         while ei==F(0):
sage:             ei=F.random_element()
sage:         e[i]=ei
sage:     return e
```

Next is the function rowreduce. As input it takes a matrix PhiM over $\mathbb{F}_q[x]$ an apply row reductions outlined in Section 4 on it. Besides the reduced matrix, it will also return the index of the row with leading position 0. Throughout the code will the list LPlist consits of the leading positions of the rows in PhiM. The list counts consits of two numbers: the first is the most common element in LPlist, and the second is how many times it appears in LPlist. The list Ilist contains all indices of rows with the most common leading position. From this list where find the index ismall which row have smallest degree.

```
sage: from collections import Counter
sage: def rowreduce(PhiM):
sage:     LPlist=[0 for i in range(0,l+1)]
sage:     for i in range(0,l+1):
sage:         maxdegree=max(PhiM[i]).degree()
sage:         for j in range(0,l+1):
sage:             if PhiM[i,j].degree()==maxdegree:
sage:                 LPlist[i]=j
sage:
sage:     counts=Counter(LPlist).most_common(1)[0]
sage:
sage:     while counts[1]>1:
sage:
sage:         Ilist =[i for i in range(0,l+1) if (LPlist[i]==counts[0])]
sage:         Imindeg=min(PhiM[i,counts[0]] for i in Ilist).degree()
sage:
sage:         for i in Ilist:
sage:             if PhiM[i,counts[0]].degree()==Imindeg:
sage:                     ismall=i
sage:         Ilist.remove(ismall)
sage:
sage:         for i in Ilist:
sage:             delta=PhiM[i,counts[0]].degree()
sage:                 -PhiM[ismall,counts[0]].degree()
sage:             alpha=PhiM[i,counts[0]].leading_coefficient()/
sage:                 PhiM[ismall,counts[0]].leading_coefficient()
sage:             PhiM[i]=PhiM[i]-alpha*x^delta*PhiM[ismall]
sage:
sage:         LPlist=[0 for i in range(0,l+1)]
sage:         for i in range(0,l+1):
```

```
sage:             maxdegree=max(PhiM[i]).degree()
sage:             for j in range(0,l+1):
sage:                 if PhiM[i,j].degree()==maxdegree:
sage:                     LPlist[i]=j
sage:         counts=Counter(LPlist).most_common(1)[0]
sage:     for i in range(0,l+1):
sage:         if LPlist[i]==0:
sage:             j=i
sage:     return([PhiM,j])
```

Next is the function Storjohann based on Algorithm 3.

```
sage: def Storjohann(Slist,Glist,nu,wlist):
sage:   l=len(Slist)
sage:    Smatrix=matrix(Slist)
sage:    Gmatrix=diagonal_matrix(Glist)
sage:    M=block_matrix([[1,matrix(Slist)],[0,Gmatrix]],subdivide=False)
sage:    PhiM=matrix(PolyR,l+1,l+1)
sage:    for i in range(0,l+1):
sage:        for j in range(0,l+1):
sage:            PhiM[i,j]=x^(wlist[j])*M[i,j].subs(x=x^nu)
sage:    reducedPhi=rowreduce(PhiM)
sage:
sage:    Phisolution=reducedPhi[0][reducedPhi[1]]
sage:
sage:    solution=[PolyR(x^(-wlist[j]/nu)*Phisolution[j].subs(x=x^(1/nu)))
sage:        for j in range(0,l+1)]
sage:    solutionmonic=[p/solution[0].leading_coefficient()
sage:        for p in solution]
sage:    return(solutionmonic)
```

Next is the function powerdecode. It takes a recieved message $r \in \mathbb{F}_q^n$ and use the power decoding explained in section 3. It will use Storjohann to solve the linearised power key equation.

```
sage: def powerdecode(r):
sage:   Rlist=[PolyR.lagrange_polynomial([(alpha[i],r[i]^t)
sage:       for i in range(0,n)]) for t in range(1,l+1)]
sage:   Glist=[G for t in range(1,l+1)]
sage:   nu=1
sage:   wlist=[l*(k-1)+1]+[(l-t)*(k-1) for t in range(1,l+1)]
sage:   solution=Storjohann(Rlist,Glist,nu,wlist)
sage:
sage:   (fhat,rem)=(solution[1]).quo_rem(solution[0])
sage:   declaredfailure=False
sage:   if rem !=0:
sage:       declaredfailure=True
sage:   else:
sage:       for i in range(1,l+1):
sage:           rem1=(solution[0]*fhat^i).quo_rem(G)[1]
sage:           rem2=(solution[0]*Rlist[i-1]).quo_rem(G)[1]
sage:           if rem1 != rem2:
sage:               declaredfailure=True
sage:   if declaredfailure:
```

```
sage:         return("declared failure")
sage:     else:
sage:         mhat=poltovec(fhat,k)
sage:         return(mhat)
```

The function simulate takes the input 3 numbers epsilon, l, and iterations. It will simulate $N =$ iterations experiments, where in each experiment it will decode an uniformly generated vector $e$ of weight epsilon using power decoding, and see whether we succeed, fail and declare failure. The three numbers it returns is: The frequency of failure $P_f =$ iterations, the frequency of error $P_e =$ iterations, and the frequency of success $P_s =$ iterations

```
sage: def simulate(epsilon,l,iterations):
sage:     m=vector(F,[0 for i in range(0,k)])
sage:     c=encode(m)
sage:     failuretimes=0
sage:     errortimes=0
sage:     for i in range(0,iterations):
sage:         e=makeerror(epsilon)
sage:         r=c+e
sage:         mhat=powerdecode(r)
sage:         if mhat!=m:
sage:             failuretimes=failuretimes+1
sage:             if mhat!="declared failure":
sage:                 errortimes=errortimes+1
sage:     failureratio=RR(failuretimes/iterations)
sage:     errorratio=RR(errortimes/iterations)
sage:     successratio=1-failureratio
sage:     return(failureratio,errorratio,successratio)
```