A Partial-Inverse Approach to Decoding Reed-Solomon Codes and Polynomial Remainder Codes

A dissertation submitted to
ETH Zurich
for the degree of
Doctor of Sciences

presented by

Jiun-Hung Yu

Master of Science in Communication Engineering,
NCTU, Hinchu, Taiwan
born on March 16, 1979
citizen of Taiwan

accepted on the recommendation of
Prof. Dr. Hans-Andrea Loeliger, examiner
Prof. Dr. Martin Bossert, co-examiner

2014
To Liya
Acknowledgments

My deepest gratitude goes to my supervisor Prof. Hans-Andrea Loeliger. Thank him for giving me the opportunity to work under his supervision, and thanks for his wise guidance on my research. I will never forget the lots of time he spent on me teaching me how to write a paper and how to precisely and clearly express an idea. What I benefited most is his rigorous logic on verifying the correctness of results, and in particular, his extremely high standard on requiring things to be perfect. Those meetings and discussions with him were so enjoyable. Moreover, I am deeply grateful for his generous support and constant encouragement.

I would like to thank Prof. Martin Bossert for kindly accepting to be co-examiner of this thesis. His work on the decoding of Reed-Solomon codes beyond half the minimum distance has inspired very much the last chapter of this thesis.

The Signal and Information Processing Laboratory is such a lovely place for doing research, where I had and have the many kind and excellent colleagues—Michèle Wigger, Stéphane Tinguely, Tobias Koch, Maja Ostojic, Ligong Wang, Murti Devarakonda, Medhi Molkaraie, Christoph Reller, Jonas Biveroni, Lukas Bolliger, Georg Wilckens, Christoph Bunte, Lukas Bruderer, Christian Schürch, Sarah Neff, Nour Zalmai, Annina Bracher, Federico Wadehn, Reto Wildhaber, and Hampus Malmberg. Thank you all and also Stefan Moser and Amos Lapidoth for the nice time of the many chats, lunches, isi dinners, isi hiking days, and the badminton time that started from Nour. Moreover, I would like to thank Bernadette Roeoesli, Rita Hildebrand, and Patrik Strebel for their help during these years.

Studying abroad has been my dream since my undergraduate days. The dream could not come true without the constant encouragement and support from Liya, my wife. I was once quite a bit irresolute whether to quit the well-paid job and make my dream come true. Liya’s words made
Just to do it, I don’t want to hear your regret in the future. In the end of 2008, Liya left her beloved job and came with me to Switzerland. My never forgetting life in Switzerland cannot be so wonderful without her company and the later coming of Guan-Ting. I certainly owe her an immense gratitude.

Moreover, I would like to thank my parents for letting me pursue my dream, without whom nothing would have been possible.

Finally, I would like to thank Prof. Yu-Ted Su, my master thesis supervisor, for his help and advices on my applying for PhD study abroad.
Abstract

The thesis develops a new approach to the central themes of algebraic coding theory. The focus here is the newly introduced concept of a partial-inverse polynomial in a quotient ring $F[x]/m(x)$. In particular, the decoding of Reed-Solomon codes can be attributed to the computation of a partial-inverse polynomial.

The problem of practical computation of a partial-inverse polynomial is closely related to the problem of shift-register synthesis, which is based on the well-known Berlekamp-Massey algorithm.

A major result of this work is a (new) algorithm for computing a partial-inverse polynomial. The new algorithm is very similar to the Berlekamp-Massey algorithm, but it is applicable generally, e.g., to extended Reed-Solomon codes and polynomial remainder codes. The algorithm can also be easily transformed into the so-called Euclidean algorithm, and thus provides a new derivation of the later.

For decoding Reed-Solomon codes, the algorithm can be directly applied to the classical key equation; however, mathematically natural is the application to a new key equation that applies in particular to generalizations of Reed-Solomon codes. Two new interpolation are also presented to accompany this new key equation.

Another focus of this work is the polynomial remainder codes, a natural generalization of Reed-Solomon codes. The theory of such codes is carefully constructed as in earlier work. In particular, varying degrees of remainders are allowed, resulting in two different definitions of the distance between two codewords. The decoding of these codes leads directly to the mentioned new key equation.

A focus of the recent algebraic coding theory is the decoding of errors beyond half the minimum distance. A mainline of such algorithms is based on generalization of the Berlekamp-Massey algorithm on several parallel sequences. In this work, a corresponding generalization of the
decoding algorithm via some partial-inverse polynomials is also developed.

**Keywords:** Error-Correcting Codes, Reed-Solomon Codes, Algebraic Coding Theory, Polynomial Remainder Codes, Berlekamp-Massey Algorithm, Padé Approximation, Euclidean Algorithm, Partial-Inverse Problem, Simultaneous Partial-Inverse Problem.
Kurzfassung

Die Arbeit entwickelt einen neuen Zugang zu zentralen Themen der algebraischen Codierungstheorie. Im Mittelpunkt steht der hier neu eingeführte Begriff eines teilinversen Polynoms in einem Restklassenring $F[x]/m(x)$. Insbesondere kann die Decodierung von Reed-Solomon Codes auf die Berechnung eines teilinversen Polynoms zurückgeführt werden. Das Problem der praktischen Berechnung eines teilinversen Polynoms ist eng verwandt mit dem Problem der Schieberegister-Synthese, welches dem bekannten Berlekamp-Massey-Algorithmus zugrunde liegt.


Zur Decodierung von Reed-Solomon Codes kann der neue Algorithmus direkt auf die klassische Schlüsselgleichung angewendet werden; mathematisch natürlicher ist aber die Anwendung auf eine neue Schlüsselgleichung, die insbesondere auch für Verallgemeinerungen von Reed-Solomon Codes gilt. Passend zu dieser neuen Schlüsselgleichung werden auch zwei neue Interpolationsformeln vorgestellt.

Ein weiterer Schwerpunkt dieser Arbeit sind polynomiale Restklassen-Codes, eine natürliche Verallgemeinerung von Reed-Solomon Codes. Die Theorie solcher Codes wird sorgfältiger aufgebaut als in früheren Arbeiten. Insbesondere werden auch Residuen von unterschiedlichem Grad zugelassen, was zu zwei unterschiedlichen Definitionen der Distanz zwischen zwei Codewörtern führt. Die Decodierung solcher Codes führt direkt zur erwähnten neuen Schlüsselgleichung.

Ein Schwerpunkt der neueren algebraischen Codierungstheorie ist

**Stichworte:** Fehlerkorrigierende Codes, Reed-Solomon Codes, algebraische Codierungstheorie, Restklassen-Codes, Berlekamp-Massey-Algorithmus, Padé-Approximation, Euklidischer Algorithmus.
Contents

Abstract vii

Kurzfassung ix

1 Introduction 1

1.1 Background and Motivation . . . . . . . . . . . . . . . . . 1
1.2 Overview of The Thesis . . . . . . . . . . . . . . . . . . . 2
1.3 How to Read The Thesis . . . . . . . . . . . . . . . . . . . 4
1.4 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . 5

2 Partial-Inverse mod m(x) and Reed-Solomon Decoding 7

2.1 The Partial-Inverse Problem . . . . . . . . . . . . . . . . . 7
2.2 Remarks On The Problem . . . . . . . . . . . . . . . . . . 9
2.3 The Algorithm to Solve The Problem . . . . . . . . . . . . 13
2.4 Two Variations of The Algorithm . . . . . . . . . . . . . . 14
  2.4.1 The Quotient-Saving Algorithm . . . . . . . . . . . . 15
  2.4.2 The Remainder-Saving Algorithm . . . . . . . . . . 17
2.5 Application to Decoding Reed-Solomon Codes . . . . . . . 19
  2.5.1 Reed-Solomon Codes . . . . . . . . . . . . . . . . . . 19
  2.5.2 Error-locator and Error Locator-based Interpolation 20
  2.5.3 Decoding With The Alternative Key Equation . . . 21
  2.5.4 Decoding With The Transformed Key Equation . . . 23
  2.5.5 Concluding Remarks: Berlekamp-Massey-Like Com-
    putation for any m(x) . . . . . . . . . . . . . . . . . . . . 25
2.6 Key Elements of The Proof . . . . . . . . . . . . . . . . . . 25
2.7 Proof of The Algorithm . . . . . . . . . . . . . . . . . . . 27
  2.7.1 Correctness of the Proposed Algorithm . . . . . . . 27
  2.7.2 Proving Propositions 2.5 and 2.6 . . . . . . . . . . 29
  2.7.3 Proof of Proposition 2.7 . . . . . . . . . . . . . . . 29
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8 Application to Padé Approximation</td>
</tr>
<tr>
<td>2.9 Application to Solving Standard Key Equation</td>
</tr>
<tr>
<td>2.10 Proving The Transformed Key Equation and Deriving Standard Key Equation</td>
</tr>
<tr>
<td>2.10.1 Deriving and Proving Theorem 2.4</td>
</tr>
<tr>
<td>2.10.2 A Way to Derive the Standard Key Equation</td>
</tr>
<tr>
<td>2.11 Constrained Partial-Inverse Problem</td>
</tr>
<tr>
<td>2.12 Joint Errors-and-Erasures Decoding of Reed-Solomon Codes</td>
</tr>
<tr>
<td>2.13 Conclusion</td>
</tr>
<tr>
<td>3 Chinese Remainder and Polynomial Remainder Codes</td>
</tr>
<tr>
<td>3.1 Introduction</td>
</tr>
<tr>
<td>3.2 Chinese Remainder Codes</td>
</tr>
<tr>
<td>3.2.1 Chinese Remainder Theorem and Codes</td>
</tr>
<tr>
<td>3.2.2 Interpolation</td>
</tr>
<tr>
<td>3.2.3 Hamming Distance and Singleton Bound</td>
</tr>
<tr>
<td>3.3 Polynomial Remainder Codes (PRC)</td>
</tr>
<tr>
<td>3.3.1 Definition and Some Examples</td>
</tr>
<tr>
<td>3.3.2 Degree-weighted Distance</td>
</tr>
<tr>
<td>3.3.3 Interpolation and Erasures Decoding</td>
</tr>
<tr>
<td>3.3.4 Minimum-Distance Decoding</td>
</tr>
<tr>
<td>3.3.5 Summary of Code Parameters</td>
</tr>
<tr>
<td>3.4 Error Factor Polynomial and The Key Equations</td>
</tr>
<tr>
<td>3.5 Error Factor-Based Interpolation</td>
</tr>
<tr>
<td>3.6 Decoding PRC via The Alternative Key Equation</td>
</tr>
<tr>
<td>3.6.1 Errors-only Decoding</td>
</tr>
<tr>
<td>3.6.2 Joint Errors-and-Erasures Decoding</td>
</tr>
<tr>
<td>3.7 Conclusion</td>
</tr>
<tr>
<td>4 GCD-Based Decoding of Polynomial Remainder Codes</td>
</tr>
<tr>
<td>4.1 An Extended GCD Algorithm</td>
</tr>
<tr>
<td>4.2 Modifications for Partially Known $E(x)$</td>
</tr>
<tr>
<td>4.3 Alternative Modifications for Partially Known $E(x)$</td>
</tr>
<tr>
<td>4.4 Summary of Decoding</td>
</tr>
<tr>
<td>4.5 Relation to Prior Work</td>
</tr>
<tr>
<td>4.6 Conclusion</td>
</tr>
<tr>
<td>4.7 An Extension</td>
</tr>
<tr>
<td>4.8 Proof of Theorem 4.3</td>
</tr>
<tr>
<td>4.9 Proof of Theorem 4.4</td>
</tr>
<tr>
<td>Chapter</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background and Motivation

Reed-Solomon codes [1] are still one of the most important class of algebraic codes nowadays. The ubiquity of Reed-Solomon codes both in theory and in practice may be attributed to the several advantages. First, they are linear codes with optimality in maximum distance separation. Secondly, they possess excellent random error and burst error correcting capabilities, and can be used as building blocks to construct new codes. Thirdly, their encoding is very simple, and in particular, there exist efficient decoding algorithms.

Among the many decoding algorithms, Berlekamp-Massey algorithm [2,3] is generally recognized as the most practical algorithm until now. The success of the Berlekamp-Massey algorithm also makes shift-register synthesis a promising perspective to generalizations of such codes and to other algebraic codes of similar sort. However, many algebraic codes are essentially constructed from polynomials, e.g., Reed-Solomon codes, and can be naturally described or better explained as polynomials. The sequence-generating viewpoint of shift-register synthesis may thus cause difficulty on the connection to decoding these polynomial codes, or may fit naturally only to the codes with restrictions, e.g., requiring the codes to be cyclic (otherwise, additional work is needed for the connection).

This motivates the thesis to develop a decoding approach that works entirely on polynomials, and fits naturally for more general class of codes, and in particular equips with efficient algorithms that can be as efficient as the Berlekamp-Massey algorithm.
1.2 Overview of The Thesis

The thesis aims to develop a natural and unified, and yet practical approach to algebraic decoding of algebraic codes.

The focus here is on decoding Reed-Solomon codes and generalizations of such codes\(^1\) from a (new) partial-inverse perspective, which begins with the problem

**Partial-Inverse Problem:** Let \(b(x)\) and \(m(x)\) be nonzero polynomials over some field \(F\), with \(\deg b(x) < \deg m(x)\). Find a nonzero polynomial \(\Lambda(x) \in F[x]\) of the smallest degree such that

\[
\deg \left( b(x)\Lambda(x) \mod m(x) \right) < d
\]

for given \(d \in \mathbb{Z}\) with \(1 \leq d \leq \deg m(x)\).

The partial-inverse problem has always a unique solution, up to a scale factor. (The solution \(\Lambda(x)\) turns out to be a minimal partial inverse that will be defined in Chapter 2.)

The partial-inverse problem includes computing inverses in \(F[x]/m(x)\) and includes Padé Approximation as a special case. In particular, the classical key equation for decoding Reed-Solomon codes is also a special case of the partial-inverse problem.

A major result of this work is a new algorithm for solving the partial-inverse problem. The new algorithm, presented in Chapter 2, is very similar to the Berlekamp-Massey algorithm, but the new algorithm works for general \(m(x)\), resulting in the natural application to a new key equation for generalizations of Reed-Solomon codes.

Two new interpolation, as the last step of decoding, are also presented to accompany this new key equation. The two interpolation can also be applied directly to erasure decoding of such codes and of their generalizations, resulting in a new and complete approach to decoding Reed-Solomon codes.

Another focus of the work is the polynomial remainder codes revisited in Chapter 3, which are a large class of codes derived from the Chinese remainder theorem that include Reed-Solomon codes as a special case. The theory of such codes is carefully constructed as in earlier work. In particular, the notions of Hamming weight and Hamming distance are generalized to the notions of degree weight and degree-weighted distance;

\(^1\)The results of this thesis were presented in part at ISIT 2011 [4], ISIT 2013 [5], and Allerton 2014 [6], and disclosed in part in [7] and [8].
moreover, code symbols of varying degrees are allowed, which are in
general polynomials of different degrees obtained from moduli of varying
degrees. The later generalization enables to lengthen a Reed-Solomon
code (by adding extra symbols of higher-degree) without increasing the
size of the underlying finite field.

The mentioned key equation is also generalized accordingly, enabling
the partial-inverse approach to apply also naturally to polynomial re-
mainder codes. Of particular interest are the generalizations of the two
mentioned interpolation, both of which now work also for Chinese re-
mainder codes.

As a subsidiary topic, gcd-based decoding of polynomial remainder
codes is also investigated in order to deal with the issue concerning the
implicit assumption of irreducibility of moduli in the prior work. This
part is elaborated in Chapter 4. The explicit allowance of reducible moduli
here also complements the prior work in the corresponding respect.

A focus of the recent algebraic coding theory is the decoding of errors
beyond half the minimum distance. A mainline of such algorithms is
based on generalization of the Berlekamp-Massey algorithm on several
parallel sequences. In this work, a corresponding generalization of the
decoding to multi-polynomial setting is also developed.

A major result here is a new algorithm that is capable of decoding
Reed-Solomon codes and interleaved Reed-Solomon codes (also known
as subfield-evaluation codes) beyond half the minimum distance. In
particular, the success of the decoding hinges on a new key equation
that amounts to the problem.

Simultaneous Partial-Inverse (SPI) Problem: For \(i = 1, 2, \ldots, L\),
let \(b^{(i)}(x)\) and \(m^{(i)}(x)\) be nonzero polynomials over some field \(F\) with
\(\deg b^{(i)}(x) < \deg m^{(i)}(x)\). The problem is to find a nonzero polynomial
\(\Lambda(x) \in F[x]\) of the smallest degree such that

\[
\deg \left( b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x) \right) < \tau^{(i)}
\]

for given \(\tau^{(i)} \in \mathbb{Z}\) with \(1 \leq \tau^{(i)} \leq \deg m^{(i)}(x)\).

A new algorithm that solves the simultaneous partial-inverse problem
is presented in Chapter 5. The SPI problem has always a unique solution
for any \(b^{(i)}(x)\), \(m^{(i)}(x)\), and any \(L \geq 1\), and the algorithm is provably
to find such a solution, up to a scale factor.

The simultaneous partial-inverse problem is clearly a nontrivial gen-
eralization of the mentioned partial-inverse problem, so is the corre-
sponding algorithm as well as the corresponding key equation; in partic-
ular, the two mentioned interpolation are also applicable here, resulting
in a unified approach to decoding Reed-Solomon codes and interleaved
Reed-Solomon codes beyond half the minimum distance.

1.3 How to Read The Thesis

The thesis consists of the three parts. Part 1 including only Chapter 2
focuses on the newly introduced partial-inverse problem and its applica-
tion to decoding Reed-Solomon codes. Part 2 including both Chapters 3
and 4 generalizes the approach of Part 1 to polynomial remainder codes.
Part 3 comprises Chapters 5–7, which extends the approach of Part 1 to
a multi-polynomial setting and focuses on the decoding of Reed-Solomon
codes and the decoding of interleaved Reed-Solomon codes beyond half
the minimum distance.

The three parts can be read independently, e.g., the reader inter-
ested in decoding beyond half the minimum distance may go to Part 3
directly. The reader interested in polynomial remainder codes may be-
gin with Part 2 or begins with Chapter 4. Since Part 1 establishes the
partial-inverse approach, it is, however, highly recommended to begin
with Chapter 2, where many new concepts and insights can be learned.

![Diagram](image)

**Figure 1.1:** The algorithms to solve the problem, and some related
applications in Chapter 2.
1.4 Contributions

The contributions of the thesis are summarized as follows.

Partial-Inverse Problem (Chapter 2)

The partial-inverse problem is a newly introduced problem. Moreover, the notion of a minimal partial inverse is a new concept. The partial-inverse algorithm together with the two variations of the algorithm is also new. For decoding Reed-Solomon codes, the presented alternative key equations and the two interpolations are all new. In fact, all the propositions, lemmas, and theorems in this chapter are new (unless otherwise mentioned). A main contribution here is the newly established approach that is applicable to decoding shortened, generalized, and extended Reed-Solomon codes. Moreover, the easy translation of the partial-inverse algorithm into a version of the Euclidean algorithm provides a new derivation of the later.

Polynomial Remainder Codes (Chapters 3 and 4)

A main contribution of this part is the two fixed-transform solutions—Theorems 3.2 and 3.3—to the problem of reconstructing a codeword \( c = (c_0, \ldots, c_{n-1}) \in C \) from a subset of its symbols \( c_\ell, \ell \in \{1, \ldots, n-1\} \), where \( C \) is a Chinese remainder code over \( R \) for \( R = \mathbb{Z} \) or \( R = F[x] \). Both the theorems did not appear in the standard expositions of the Chinese remainder theorem, and thus they appear to be new. Mathematically, the two theorems also contribute as a fundamental proposition subsidiary to the Chinese remainder theorem.

Another contribution is that the polynomial remainder codes are studied more carefully than in previous work. The moduli of such codes are allowed to be arbitrary polynomials provided that they are relatively prime. In particular, the notions of degree weight and degree-weighted distance are introduced, and the notion of error locator polynomial is generalized as well. If the moduli are reducible, the notion of error locator polynomial is replaced by the newly introduced notion of an error factor polynomial. With these generalizations, together with the generalization of the alternative key equation, the partial-inverse approach of Chapter 2 applies naturally to polynomial remainders codes.

The main contribution of the gcd-based decoding of Chapter 4 is that the gcd-based decoding algorithms therein explicitly work for general polynomial remainder codes that can be encoded by reducible moduli of different degrees.
Simultaneous Partial Inverses (Chapters 5–7)

A main contribution of this part is the newly introduced simultaneous partial-inverse problem and the new algorithm of Chapter 5 for solving the problem. In particular, the proof for the correctness of the algorithm is instructive and also interesting, which also serves as an alternative proof and provides yet a new insight to the partial-inverse algorithm of Chapter 2.

The main contribution of Chapter 6 is the new key equation together with the new algorithm for decoding interleaved Reed-Solomon codes; the algorithm can correct column errors beyond half the minimum distance. In particular, any error matrix $E$ with less than $n - k$ column errors can be corrected provided that the rank of $E$ equals the number of column errors.

The contribution of Chapter 7 is the demonstration of the algorithm to decoding a scheme of low rate Reed-Solomon codes beyond half the minimum distance. The resulting algorithm is a unique-decoding algorithm that returns at most one codeword. The algorithm is new, and its decoding radius coincides with the best of the known unique-decoding algorithms at the present time.
Chapter 2
Partial-Inverse mod $m(x)$
and Reed-Solomon Decoding

The partial-inverse approach to decoding Reed-Solomon codes begins with the partial-inverse problem that was mentioned in Chapter 1. The problem is recalled below and is discussed in greater detail. In particular, it will be clear how the problem includes computing inverses in $F[x]/m(x)$, computing Padé approximants, and decoding Reed-Solomon codes as special cases. A major result here is a new algorithm together with two variations of the algorithm for solving the partial-inverse problem. The algorithm is similar to the Berlekamp-Massey algorithm, but it works for general $m(x)$. Moreover, two new interpolation and a new key equation for Reed-Solomon codes are presented.

2.1 The Partial-Inverse Problem

Partial-Inverse Problem: Let $b(x)$ and $m(x)$ be nonzero polynomials over some field $F$, with $\deg b(x) < \deg m(x)$. Find a nonzero polynomial $\Lambda(x) \in F[x]$ of the smallest degree such that

$$\deg \left( b(x)\Lambda(x) \mod m(x) \right) < d$$

(2.1)

for given $d \in \mathbb{Z}$ with $1 \leq d \leq \deg m(x)$.

□
The problem has always a unique solution, up to a scale factor (see Proposition 2.1 of Section 2.2).

In the special case where \( d = 1 \) and \( \gcd(b(x), m(x)) = 1 \), the problem reduces to computing the inverse of \( b(x) \) in \( F[x]/m(x) \).

Another special case is the standard key equation for decoding Reed-Solomon codes [1–3,9–11]. In this case, \( b(x) \) is a given syndrome polynomial, and \( m(x) = x^n - k \) where \( n \) and \( k \) are the blocklength and the dimension of the code, respectively. The desired solution is a unique pair \( \Lambda(x) \) and \( r(x) = b(x)\Lambda(x) \mod m(x) \) such that \( \gcd(\Lambda(x), r(x)) = 1 \) and \( \deg r(x) < \deg \Lambda(x) \leq (n - k)/2 \). As we will see in Section 2.9 this is a Partial-Inverse Problem with \( d = \lceil (n - k)/2 \rceil \).

In fact, computing Padé approximants [12] can also be viewed as a Partial-Inverse Problem. In this respect, an immediate translation is given in Section 2.8.

The Partial-Inverse Problem is similar to the linear feedback shift-register synthesis (LFSRS) problem [3], and can be used for similar purposes, e.g., for decoding Reed-Solomon codes. However, it is not identical to LFSRS problem; e.g., the Partial-Inverse Problem has always a unique solution, but the LFSRS problem does not have this property.

The Partial-Inverse Problem with general \( m(x) \) arises from an alternative key equation for decoding Reed-Solomon codes that will be discussed in Section 2.5. This alternative key equation and the decoding algorithm generalize naturally to polynomial remainder codes, as will be discussed in Chapter 3.

The Partial-Inverse Problem can be solved by the algorithms proposed in Sections 2.3 and 2.4. In particular, we will see how the algorithms work in structure and spirit similar to the Berlekamp-Massey (BM) algorithm [2,3], but they work naturally for general \( m(x) \).

The chapter is structured as follows. Section 2.2 comprises a number of remarks on the Partial-Inverse Problem. The new algorithm PIA is proposed in Section 2.3. An extension and one variant of PIA are presented in Section 2.4. Decoding Reed-Solomon codes via the alternative key equation or the transformed key equation is described in Section 2.5. In Sections 2.6 and 2.7 we prove the proposed algorithms. In Sections 2.8 and 2.9 we show that computing Padé approximants and solving the standard key equation for decoding Reed-Solomon codes are actually special cases of the Partial-Inverse Problem. In Section 2.10 we prove the transformed key equation, and discuss the connection between the standard key equation and the alternative key equation. In Section 2.11 we introduce a constrained Partial-Inverse Problem, which
2.2 Remarks On The Problem

is also interesting; an application to joint errors-and-erasures decoding of Reed-Solomon codes is discussed in Section 2.12. Finally, in Section 2.13 we conclude the chapter.

The following notation will be used. The Hamming weight of $e \in F^n$ will be denoted by $w_H(e)$. The coefficient of $x^\ell$ of a polynomial $b(x) \in F[x]$ will be denoted $b_\ell$. The leading coefficient (i.e., the coefficient of $x^{\deg b(x)}$) of a nonzero polynomial $b(x)$ will be denoted by $\lcf b(x)$, and we also define $\lcf(0) \triangleq 0$. We will use “mod” both as in $r(x) = b(x) \mod m(x)$ (the remainder of a division) and as in $b(x) \equiv r(x) \mod m(x)$ (a congruence modulo $m(x)$). For $x \in \mathbb{R}$, $\lceil x \rceil$ is the smallest integer not smaller than $x$.

2.2 Remarks On The Problem

We begin with a number of somewhat technical remarks on the Partial-Inverse Problem as stated in Section 2.1. The reader may prefer to proceed directly to Section 2.3.

1. The stated assumptions imply $\deg m(x) \geq 1$.

2. For $d = \deg m(x)$, the problem is solved by $\Lambda(x) = 1$. Smaller values of $d$ will normally require a polynomial $\Lambda(x)$ of higher degree.

3. In the special case where $d = 1$, we have the following solutions. If $\gcd(b(x), m(x)) = 1$, then $b(x)$ has an inverse in $F[x]/m(x)$ and $\Lambda(x)$ is that inverse (up to a scale factor); otherwise, the solution is $\Lambda(x) = m(x)/\gcd(b(x), m(x))$, which yields

$$b(x)\Lambda(x) \mod m(x) = 0.$$ #4. The previous remark implies that the problem has a solution for any $d \geq 1$.

Moreover, we have

**Proposition 2.1 (Uniqueness of Solution)** The solution $\Lambda(x)$ of the Partial-Inverse Problem is unique up to a scale factor.

The proof is given below. As we will see shortly, the solution of the problem is closely related to the notion of minimal partial inverse:
Definition 2.1 (Minimal Partial Inverse) For fixed nonzero $b(x)$ and $m(x) \in F[x]$ with $\deg b(x) < \deg m(x)$, a nonzero polynomial $\Lambda(x) \in F[x]$ is a minimal partial inverse of $b(x)$ if every nonzero $\Lambda^{(1)}(x) \in F[x]$ with

$$\deg \left( b(x)\Lambda^{(1)}(x) \mod m(x) \right) \leq \deg \left( b(x)\Lambda(x) \mod m(x) \right)$$

satisfies $\deg \Lambda^{(1)}(x) \geq \deg \Lambda(x)$. □

Proposition 2.2 (Uniqueness of Minimal Partial Inverse) Every minimal partial inverse $\Lambda(x)$ (for fixed $b(x)$ and $m(x)$) is unique (up to a scale factor).

The proof is given below.

Proposition 2.3 The solution $\Lambda(x)$ of the Partial-Inverse Problem is a minimal partial inverse of $b(x)$.

Conversely, every minimal partial inverse $\Lambda(x)$ of $b(x)$ solves the Partial-Inverse Problem with

$$d = \deg \left( b(x)\Lambda(x) \mod m(x) \right) + 1.$$ (2.3)

Proof: Let $\Lambda(x)$ be the solution of the Partial-Inverse Problem. If $\Lambda(x)$ is not a minimal partial inverse, then there exists $\Lambda^{(1)}(x)$ that satisfies (2.2) but $\deg \Lambda^{(1)}(x) < \deg \Lambda(x)$, which is a contradiction. The converse is obvious. □

In fact, we will see (cf. Proposition 2.6 or Section 2.7 for more details) that the proposed algorithm can be used to compute all minimal partial inverses (for fixed $b(x)$ and $m(x)$) in a single run.

Theorem 2.1 If $\Lambda(x)$ solves the Partial-Inverse Problem, then

$$\deg \Lambda(x) \leq \deg m(x) - d.$$ (2.4)

If $\Lambda(x)$ is a minimal partial inverse of $b(x)$, then

$$\deg \Lambda(x) < \deg m(x) - \deg \left( b(x)\Lambda(x) \mod m(x) \right).$$ (2.5)

The proof will be given in Section 2.7 (The two statements in Theorem 2.1 are easily seen to be equivalent due to Proposition 2.3).
Corollary 2.1  In consequence of (2.4), coefficients $b_\ell$ of $b(x)$ with
\[ \ell < 2d - \deg m(x) \quad (2.6) \]
and coefficients $m_\ell$ of $m(x)$ with
\[ \ell < 2d - \deg m(x) + 1 \quad (2.7) \]
are irrelevant for the solution $\Lambda(x)$.

The corollary simply follows from the observation: these coefficients do not affect (2.1) since
\[ b(x)\Lambda(x) \mod m(x) = b(x)\Lambda(x) - q(x)m(x) \quad (2.8) \]
with $\deg q(x) < \deg \Lambda(x) \leq \deg m(x) - d$. Such irrelevant coefficients may be set to zero without affecting the solution $\Lambda(x)$.

Up to this point, we can easily prove the theorem.

Theorem 2.2  Let $b(x)$ and $m(x)$ be nonzero polynomials over some field $F$, with $\deg b(x) < \deg m(x)$. Then there exist $\Lambda(x) \neq 0$ and
\[ r(x) \triangleq b(x)\Lambda(x) \mod m(x) \quad (2.9) \]
such that $\deg r(x) < \deg m(x) - t$ and $\deg \Lambda(x) \leq t$ for any given $t \in \mathbb{Z}$ with $0 \leq t \leq \deg m(x)$.

Proof:  The two cases $t = 0$ and $t = \deg m(x)$ are obvious from setting $\Lambda(x) = 1$ and $\Lambda(x) = m(x)$, respectively. As for $0 < t < \deg m(x)$, the theorem follows from the first part of Theorem 2.1 with $d = \deg m(x) - t$, and from Remark 4 above: the Partial-Inverse Problem has a solution for any $d \geq 1$.

Now, we prove Proposition 2.1.

Proof of Proposition 2.1:  Let $\Lambda^{(1)}(x)$ and $\Lambda^{(2)}(x)$ be two solutions of the problem, which implies $\deg \Lambda^{(1)}(x) = \deg \Lambda^{(2)}(x) \geq 0$. Define
\[ r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \mod m(x) \quad (2.10) \]
\[ r^{(2)}(x) \triangleq b(x)\Lambda^{(2)}(x) \mod m(x) \quad (2.11) \]
and consider
\[ \Lambda(x) \triangleq \left( \lcf \Lambda^{(2)}(x) \right)\Lambda^{(1)}(x) - \left( \lcf \Lambda^{(1)}(x) \right)\Lambda^{(2)}(x). \quad (2.12) \]
Then
\[ r(x) \triangleq b(x)\Lambda(x) \mod m(x) \quad (2.13) \]
\[ = \left( \text{lcf } \Lambda^{(2)}(x) \right)r^{(1)}(x) - \left( \text{lcf } \Lambda^{(1)}(x) \right)r^{(2)}(x) \quad (2.14) \]
by the natural ring homomorphism \( F[x] \to F[x]/m(x) \). Clearly, (2.14) implies that \( \Lambda(x) \) also satisfies (2.1). But (2.12) implies \( \deg \Lambda(x) < \deg \Lambda^{(1)}(x) \), which is a contradiction unless \( \Lambda(x) = 0 \). Thus \( \Lambda(x) = 0 \), which means that \( \Lambda^{(1)}(x) \) and \( \Lambda^{(2)}(x) \) are equal up to a scale factor. \( \square \)

A similar argument proves Proposition 2.2.

**Proof of Proposition 2.2:** Assume that \( \Lambda^{(1)}(x) \) and \( \Lambda^{(2)}(x) \) are two minimal partial inverses of \( b(x) \) with \( \deg \Lambda^{(1)}(x) = \deg \Lambda^{(2)}(x) \). Define \( r^{(1)}(x) \) and \( r^{(2)}(x) \) as in (2.10) and (2.11). Then, the definition of minimal partial inverse implies that \( \deg r^{(1)}(x) = \deg r^{(2)}(x) \).

Now consider \( \Lambda(x) \) as in (2.12) and define \( r(x) \) as in (2.13), which is equal to (2.14). We then obtain \( \deg r(x) \leq \deg r^{(1)}(x) \). But (2.12) implies \( \deg \Lambda(x) < \deg \Lambda^{(1)}(x) \), which leads to a contradiction (to the definition of minimal partial inverse) unless \( \Lambda(x) = 0 \). Thus \( \Lambda^{(1)}(x) \) and \( \Lambda^{(2)}(x) \) must be equal (up to a scale factor). \( \square \)

Finally, we end the section by the proposition.

**Proposition 2.4** Let \( \Lambda(x) \in F[x] \) be a minimal partial inverse of \( b(x) \) (for fixed \( m(x) \)). Then any nonzero \( \Lambda^{(1)}(x) \in F[x] \) such that
\[ \deg \left( b(x)\Lambda^{(1)}(x) \mod m(x) \right) < \deg \left( b(x)\Lambda(x) \mod m(x) \right), \quad (2.15) \]
satisfies \( \deg \Lambda^{(1)}(x) > \deg \Lambda(x) \).

**Proof:** Assume that there exists such \( \Lambda^{(1)}(x) \) that satisfies (2.15) with \( \deg \Lambda^{(1)}(x) = \deg \Lambda(x) \). Then the polynomial
\[ \Lambda^{(2)}(x) \triangleq \left( \text{lcf } \Lambda^{(1)}(x) \right)\Lambda(x) - \left( \text{lcf } \Lambda(x) \right)\Lambda^{(1)}(x) \quad (2.16) \]
will satisfy
\[ \deg \left( b(x)\Lambda^{(2)}(x) \mod m(x) \right) \leq \deg \left( b(x)\Lambda(x) \mod m(x) \right) \quad (2.17) \]
by the same reasoning as in the proof of Proposition 2.1. But \( \Lambda^{(2)}(x) \) has degree \( \deg \Lambda^{(2)}(x) < \deg \Lambda(x) \), which contradicts the fact that \( \Lambda(x) \) is a minimal partial inverse. \( \square \)
2.3 The Algorithm to Solve The Problem

The Partial-Inverse Problem as stated in Section 2.1 can be solved by the following algorithm.

**Partial-Inverse Algorithm (PIA):**

**Input:** \( b(x), m(x), \) and \( d \) as in the problem statement.

**Output:** \( \Lambda(x) \) as in the problem statement.

1. if \( \deg b(x) < d \) begin
2.   return \( \Lambda(x) := 1 \)
3. end
4. \( \Lambda^{(1)}(x) := 0, \) \( d_1 := \deg m(x), \) \( \kappa_1 := \lcf m(x) \)
5. \( \Lambda^{(2)}(x) := 1, \) \( d_2 := \deg b(x), \) \( \kappa_2 := \lcf b(x) \)
6. loop begin
7.   \( \Lambda^{(1)}(x) := \kappa_2 \Lambda^{(1)}(x) - \kappa_1 x^{d_1 - d_2} \Lambda^{(2)}(x) \)
8.   \( d_1 := \deg (b(x)\Lambda^{(1)}(x) \mod m(x)) \)
9.   if \( d_1 < d \) begin
10.      return \( \Lambda(x) := \Lambda^{(1)}(x) \)
11.   end
12. \( \kappa_1 := \lcf (b(x)\Lambda^{(1)}(x) \mod m(x)) \)
13. if \( d_1 < d_2 \) begin
14.   \( (\Lambda^{(1)}(x), \Lambda^{(2)}(x)) := (\Lambda^{(2)}(x), \Lambda^{(1)}(x)) \)
15.   \( (d_1, d_2) := (d_2, d_1) \)
16.   \( (\kappa_1, \kappa_2) := (\kappa_2, \kappa_1) \)
17. end
18. end

Note that lines \( 14 \) and \( 16 \) simply swap \( \Lambda^{(1)}(x) \) with \( \Lambda^{(2)}(x) \), \( d_1 \) with \( d_2 \), and \( \kappa_1 \) with \( \kappa_2 \). The only actual computations are in lines 7 and 8.

The pivotal part of the algorithm is line 7, which is explained by Lemma 2.1 of Section 2.6. The correctness of this algorithm will be proved in Section 2.7. In particular, we will see that the value of \( d_1 \) is reduced in every execution of line 8.

Note that lines 8 and 12 do not require the computation of the entire polynomial \( b(x)\Lambda^{(1)}(x) \mod m(x) \). Indeed, lines 8–12 can be replaced by the following loop:

**Equivalent Alternative to Lines 8–12:**
repeat
\[ d_1 := d_1 - 1 \]
if \( d_1 < d \) begin
\[ \text{return } \Lambda(x) := \Lambda^{(1)}(x) \]
end
\[ \kappa_1 := \text{coefficient of } x^{d_1} \text{ in } b(x)\Lambda^{(1)}(x) \mod m(x) \]
until \( \kappa_1 \neq 0 \)

In the special case where \( m(x) = x^\nu \), line 36 amounts to
\[ \kappa_1 := b_{d_1}\Lambda_0^{(1)} + b_{d_1-1}\Lambda_1^{(1)} + \ldots + b_{d_1-\tau}\Lambda_{\tau}^{(1)} \]
with \( \tau \triangleq \deg \Lambda^{(1)}(x) \) and where \( b_{\ell} \triangleq 0 \) for \( \ell < 0 \). In the other special case where \( m(x) = x^n - 1 \) as in (2.24) of Section 2.5, line 36 becomes
\[ \kappa_1 := b_{d_1}\Lambda_0^{(1)} + b_{|d_1-1|}\Lambda_1^{(1)} + \ldots + b_{|d_1-\tau|}\Lambda_{\tau}^{(1)} \]
with \( b_{[\ell]} \triangleq b_{\ell \mod n} \). In both cases, the proposed algorithm looks very much like, and is as efficient as, the Berlekamp-Massey algorithm [3].

It is worth noting that in the slightly general case where \( m(x) = x^n - p(x) \) and \( \deg p(x) \leq 2d - \deg m(x) \), line 36 also amounts to line 41 because in such a case, \( p(x) \) is irrelevant as if \( m(x) = x^n \) (cf. Corollary 2.1 of Section 2.2).

We conclude the section by the two propositions regarding the minimal partial inverses of \( b(x) \) (defined in Section 2.2).

**Proposition 2.5** Both the \( \Lambda^{(1)}(x) \) and \( \Lambda^{(2)}(x) \) between lines \( \ref{line13} \) and \( \ref{line14} \) (of PIA) are minimal partial inverses of \( b(x) \) and \( \deg \Lambda^{(1)}(x) > \deg \Lambda^{(2)}(x) \).

**Proposition 2.6** For fixed nonzero polynomials \( b(x) \) and \( m(x) \) with \( \deg b(x) < \deg m(x) \), all the minimal partial inverses of \( b(x) \) can be obtained by PIA with \( d = 1 \).

The two propositions will be proved in Section 2.7. The connection between the minimal partial inverses and the Padé approximants is given in Section 2.8.

2.4 Two Variations of The Algorithm

The quotient-saving algorithm below is an extended version of PIA for solving the same Partial-Inverse Problem. The quotient-saving algorithm works for general \( m(x) \) in explicitly the same fashion (as the
2.4 Two Variations of The Algorithm

Berlekamp-Massey algorithm) as if \(m(x) = x^n\) as in Section 2.3. The remainder-saving algorithm of Section 2.4.2 is another variation of PIA, which behaves, however, in principle similar to the Euclidean algorithm.

2.4.1 The Quotient-Saving Algorithm

The idea is to iteratively compute also the quotient of the computation \(b(x)\Lambda^{(1)}(x) \mod m(x)\), cf. Proposition 2.7 below.

**Quotient-Saving Algorithm (QSA):**

**Input:** \(b(x), m(x),\) and \(d\) as in the problem statement.

**Output:** \(\Lambda(x)\) as in the problem statement.

The algorithm is exactly the algorithm of Section 2.3 with the following additions:

- Line 5: we also initialize the two polynomials
  \[
  \Gamma^{(1)}(x) := 1, \quad \Gamma^{(2)}(x) := 0
  \]

- Line 7: we also compute
  \[
  \Gamma^{(1)}(x) := \kappa_2 \Gamma^{(1)}(x) - \kappa_1 x^{d_1 - d_2} \Gamma^{(2)}(x)
  \]

- Line 14: we also do the swap
  \[
  (\Gamma^{(1)}(x), \Gamma^{(2)}(x)) := (\Gamma^{(2)}(x), \Gamma^{(1)}(x))
  \]

Again, the algorithm QSA is exactly the PIA algorithm of Section 2.3 with the respective additions in line 5, line 7, and line 14.

**Proposition 2.7** For QSA, \(b(x)\Lambda^{(1)}(x) \mod m(x)\) of line 8 (and/or line 12) amounts to

\[
b(x)\Lambda^{(1)}(x) \mod m(x) = b(x)\Lambda^{(1)}(x) + m(x)\Gamma^{(1)}(x)
\]  

(2.18)

The proposition will be proved in Section 2.7.

As the algorithm of Section 2.3 lines 8-12 of QSA can be equivalently carried out by lines 31-37 as in Section 2.3. It then follows from Proposition 2.7 that line 36 can be computed as follows

\[
\kappa_1 := \sum_{\ell=0}^{\tau} b_{d_1-\ell} \Lambda^{(1)}_{\ell} + \sum_{\ell=0}^{\nu} m_{d_1-\ell} \Gamma^{(1)}_{\ell}
\]  

(2.19)
with \( \tau \triangleq \deg \Lambda^{(1)}(x) \) and \( \nu \triangleq \deg \Gamma^{(1)}(x) \), and where both \( b_\ell \triangleq 0 \) and \( m_\ell \triangleq 0 \) for \( \ell < 0 \). We then obtain the following version of QSA, where the original lines 8–12 have been replaced by lines 31–37:

**Quotient-Saving Algorithm (QSA):**

**Input:** \( b(x), m(x), \) and \( d \) as in the problem statement.

**Output:** \( \Lambda(x) \) as in the problem statement.

```plaintext
1 if \( \deg b(x) < d \) begin
2 return \( \Lambda(x) := 1 \)
3 end
4 \( \Lambda^{(1)}(x) := 0, \ d_1 := \deg m(x), \ \kappa_1 := \lcf m(x) \)
5 \( \Lambda^{(2)}(x) := 1, \ d_2 := \deg b(x), \ \kappa_2 := \lcf b(x) \)
6 \( \Gamma^{(1)}(x) := 1, \ \Gamma^{(2)}(x) := 0 \)
7 loop begin
8 \( \Lambda^{(1)}(x) := \kappa_2 \Lambda^{(1)}(x) - \kappa_1 x^{d_1-d_2} \Lambda^{(2)}(x) \)
9 \( \Gamma^{(1)}(x) := \kappa_2 \Gamma^{(1)}(x) - \kappa_1 x^{d_1-d_2} \Gamma^{(2)}(x) \)
10 repeat
11 \( d_1 := d_1 - 1 \)
12 if \( d_1 < d \) begin
13 return \( \Lambda(x) := \Lambda^{(1)}(x) \)
14 end
15 \( \kappa_1 := \sum_{\ell=0}^\tau b_\ell x^{d_1-\ell} \Lambda^{(1)}_\ell + \sum_{\ell=0}^\nu m_\ell x^{d_1-\ell} \Gamma^{(1)}_\ell \)
16 until \( \kappa_1 \neq 0 \)
17 if \( d_1 < d_2 \) begin
18 \( (\Lambda^{(1)}(x), \Lambda^{(2)}(x)) := (\Lambda^{(2)}(x), \Lambda^{(1)}(x)) \)
19 \( (\Gamma^{(1)}(x), \Gamma^{(2)}(x)) := (\Gamma^{(2)}(x), \Gamma^{(1)}(x)) \)
20 \( (d_1, d_2) := (d_2, d_1) \)
21 \( (\kappa_1, \kappa_2) := (\kappa_2, \kappa_1) \)
22 end
23 end
```

It is clear that QSA in such a form indeed looks very much like, and is as efficient as, the Berlekamp-Massey algorithm [3], but QSA works for general \( m(x) \).

For all the special cases as mentioned in Section 2.3, \( \Gamma^{(1)}(x) \) and \( \Gamma^{(2)}(x) \) of lines 5, 7, and 14 of QSA can all be removed; QSA then reduces to the corresponding PIA of Section 2.3.
2.4.2 The Remainder-Saving Algorithm

We now show an easy modification that turns the PIA of Section 2.3 into a (inversionless) variant of Euclidean algorithm. The idea here is to extend the PIA to iteratively compute the remainder of \( b(x)\Lambda^{(1)}(x) \mod m(x) \), i.e., computing the remainder of \( b(x)\Lambda^{(1)}(x) \) divided by \( m(x) \).

Indeed, it is easy seen from Lemma 2.1 and (2.59) of Section 2.6 that \( b(x)\Lambda^{(1)}(x) \mod m(x) \) of line 8 of PIA can actually be computed iteratively throughout the algorithm as follows.

**Remainder-Saving Algorithm (RSA):**

**Input:** \( b(x) \), \( m(x) \), and \( d \) as in the problem statement.

**Output:** \( \Lambda(x) \) as in the problem statement.

1. \( \text{if } \deg b(x) < d \text{ begin} \)
2. \( \text{return } \Lambda(x) := 1 \)
3. \( \text{end} \)
4. \( \Lambda^{(1)}(x) := 0, \ d_1 := \deg m(x) , \ \kappa_1 := \operatorname{lcf} m(x) \)
5. \( \Lambda^{(2)}(x) := 1, \ d_2 := \deg b(x) , \ \kappa_2 := \operatorname{lcf} b(x) \)
6. \( r^{(1)}(x) := m(x), \ r^{(2)}(x) := b(x) \)
7. \( \text{loop begin} \)
8. \( \Lambda^{(1)}(x) := \kappa_2\Lambda^{(1)}(x) - \kappa_1 x^{d_1 - d_2}\Lambda^{(2)}(x) \)
9. \( r^{(1)}(x) := \kappa_2 r^{(1)}(x) - \kappa_1 x^{d_1 - d_2} r^{(2)}(x) \)
10. \( \text{end} \)
11. \( d_1 := \deg r^{(1)}(x) \)
12. \( \text{if } d_1 < d \text{ begin} \)
13. \( \text{return } \Lambda(x) := \Lambda^{(1)}(x) \)
14. \( \text{end} \)
15. \( \kappa_1 := \operatorname{lcf} r^{(1)}(x) \)
16. \( \text{if } d_1 < d_2 \text{ begin} \)
17. \( (\Lambda^{(1)}(x), \Lambda^{(2)}(x)) := (\Lambda^{(2)}(x), \Lambda^{(1)}(x)) \)
18. \( (r^{(1)}(x), r^{(2)}(x)) := (r^{(2)}(x), r^{(1)}(x)) \)
19. \( (d_1, d_2) := (d_2, d_1) \)
20. \( (\kappa_1, \kappa_2) := (\kappa_2, \kappa_1) \)
21. \( \text{end} \)
22. \( \text{end} \)

The algorithm RSA is derived from the PIA of Section 2.3 by adding the following additions into PIA followed by a modification:
Partial-Inverse mod $m(x)$ and Reed-Solomon Decoding

- Line 5: we also initialize the two polynomials
  \[ r^{(1)}(x) := m(x), \quad r^{(2)}(x) := b(x) \]

- Line 7: we also compute
  \[ r^{(1)}(x) := \kappa_2 r^{(1)}(x) - \kappa_1 x^{d_1 - d_2} r^{(2)}(x) \]

- Line 14: we also do the swap
  \[(r^{(1)}(x), r^{(2)}(x)) := (r^{(2)}(x), r^{(1)}(x))\]

- Replace everywhere $b(x) \Lambda^{(1)}(x) \mod m(x)$ by $r^{(1)}(x)$

The RSA, although it is not immediately obvious, turns out to be a version of the standard Euclidean algorithm (cf. Section 4.1 of Chapter 4) for computing the gcd of $b(x)$ and $m(x)$. In this respect, run RSA with $d = 0$; we then obtain $r^{(2)}(x) = \gamma \gcd(b(x), m(x))$ (for some nonzero scale factor $\gamma$) when RSA stops.

Running RSA with $d = 1$ offers another efficient way to compute the inverse of $b(x)$ in $F[x]/m(x)$ if it exists. In contrast to the standard Euclidean algorithm, there is no scalar inversion involved for doing the division.

Relation to Prior Work

The PIA of Section 2.3 and also QSA, as they stand, are “computationally” very similar to the Berlekamp-Massey algorithm [2,3]. On the other hand, the algorithm RSA, as a variant of PIA, works in principle similar to (or say in principle the same as) Euclidean algorithm. In fact, our translation of the PIA into RSA provides a different perspective and complements the prior work [13–16] on the connection of the Berlekamp-Massey algorithm and the Euclidean algorithm: Dornstetter [13] derived from the extended Euclidean algorithm a simplified form that works in a principle similar to the Berlekamp-Massey algorithm, and also argued that both the algorithms produce the same “partial” results (up to a reciprocation and a scale factor). The same approach was adopted in [15] and [16] in a more clear way to make the simplified Euclidean algorithm computationally more resemble the Berlekamp-Massey algorithm. While in [14], their equivalence is argued by means of the fundamental iterative algorithm (that processes the syndrome matrix obtained from the standard key equation); in [17] and [18], the authors argued the equivalence by means of continued fractions.
2.5 Application to Decoding Reed-Solomon Codes

Decoding Reed-Solomon codes (up to half the minimum distance) can be reduced rather directly to the Partial-Inverse Problem of Section 2.1 via solving either the alternative key equation or the transformed key equation below.

2.5.1 Reed-Solomon Codes

Let $F$ be a finite field, let $\beta_0, \ldots, \beta_{n-1}$ be $n$ different elements of $F$, let

$$m(x) \triangleq \prod_{\ell=0}^{n-1} (x - \beta_{\ell}),$$

(2.20)

let $F[x]/m(x)$ be the ring of polynomials modulo $m(x)$, and let $\psi$ be the evaluation mapping

$$\psi : F[x]/m(x) \to F^n : a(x) \mapsto (a(\beta_0), \ldots, a(\beta_{n-1})),
\tag{2.21}$$

which is a ring isomorphism\footnote{A Reed-Solomon code with blocklength $n$ and dimension $k$ may be defined as

$$\{c = (c_0, \ldots, c_n) \in F^n : \deg \psi^{-1}(c) < k\},$$

(2.22)

usually with the additional condition that

$$\beta_{\ell} = \alpha^{\ell} \quad \text{for } \ell = 0, \ldots, n - 1,$$

(2.23)

where $\alpha \in F$ is a primitive $n$-th root of unity. The condition (2.23) implies

$$m(x) = x^n - 1,$$

(2.24)

and turns $\psi$ into a discrete Fourier transform \cite{10}. However, (2.23) and (2.24) will not be required below.

Note also that the condition

$$\beta_{\ell} \neq 0 \text{ for all } \ell \in \{0, \ldots, n - 1\}.$$  

(2.25)

is not required either.

\footnote{The inverse mapping $\psi^{-1}$ can be carried out, for example, by Lagrange interpolation or by (3.2) since (2.21) is a special case of (3.1).}
2.5.2 Error-locator and Error Locator-based Interpolation

Let \( y = (y_0, \ldots, y_{n-1}) \in \mathbb{F}^n \) be the received word, which we wish to decompose into
\[
y = c + e \quad (2.26)
\]
where \( c \in \mathcal{C} \) is a codeword and where the Hamming weight of \( e = (e_0, \ldots, e_{n-1}) \in \mathbb{F}^n \) is as small as possible.

Let \( C(x) \triangleq \psi^{-1}(c) \), and analogously \( E(x) \triangleq \psi^{-1}(e) \) and \( Y(x) \triangleq \psi^{-1}(y) \). Clearly, we have \( \deg C(x) < k \) and \( \deg E(x) < \deg m(x) = n \).

For any \( e \in \mathbb{F}^n \), we define the error locator polynomial
\[
\Lambda_e(x) \triangleq \prod_{\ell \in \{0, \ldots, n-1\}} (x - \beta^\ell). \quad (2.27)
\]
Clearly, \( \deg \Lambda_e(x) = w_H(e) \) and
\[
E(x) \Lambda_e(x) \mod m(x) = 0. \quad (2.28)
\]

The following two propositions can be used to complete decoding as will be discussed in Section 2.5.3, both of which also apply to erasures-only decoding.

**Proposition 2.8** If \( \Lambda(x) \) is a nonzero multiple of \( \Lambda_e(x) \) with \( \deg \Lambda(x) \leq n - k \), then
\[
C(x) = \frac{Y(x) \Lambda(x) \mod m(x)}{\Lambda(x)} \quad (2.29)
\]

**Proof:** If \( \Lambda(x) \) has the stated properties, then
\[
Y(x) \Lambda(x) \mod m(x) = C(x) \Lambda(x) \mod m(x) + E(x) \Lambda(x) \mod m(x) \quad (2.30)
\]
\[
= C(x) \Lambda(x), \quad (2.31)
\]
where the second term in (2.30) vanishes because of (2.28). \( \square \)

**Proposition 2.9** If \( \Lambda(x) = \gamma \Lambda_e(x) \) for some nonzero \( \gamma \in \mathbb{F} \) with \( \deg \Lambda(x) \leq n - k \), then
\[
C(x) = Y(x) \mod m_S(x) \quad (2.32)
\]
where \( m_S(x) \triangleq m(x)/\Lambda(x) \).
Proof: Note that \( m_S(x) \) has degree \( \deg m_S(x) \geq k > \deg C(x) \). Note also that \( m_S(x) \) divides gcd\((E(x), m(x))\). We then have

\[
Y(x) \mod m_S(x) = C(x) + E(x) \mod m_S(x) = C(x).
\]

\[\Box\]

2.5.3 Decoding With The Alternative Key Equation

The error-locator polynomial (2.27) can be found by solving the alternative key equation.

**Theorem 2.3 (Alternative Key Equation)** If \( w_H(e) \leq \frac{n-k}{2} \), then the error locator polynomial \( \Lambda_e(x) \) satisfies

\[
\deg \left( (Y(x)\Lambda_e(x) \mod m(x)) \right) < \frac{n+k}{2}
\]

(2.35)

Conversely, for any \( y \) and \( e \in F^n \) and \( t \in \mathbb{R} \) with

\[
w_H(e) \leq t \leq \frac{n-k}{2},
\]

(2.36)

if some nonzero \( \Lambda(x) \in F[x] \) with \( \deg \Lambda(x) \leq t \) satisfies

\[
\deg (Y(x)\Lambda(x) \mod m(x)) < n - t,
\]

(2.37)

then \( \Lambda(x) \) is a multiple of \( \Lambda_e(x) \).

The proof is given below.

**Corollary 2.2** For any \( y \) and \( e \in F^n \), if \( w_H(e) \leq \frac{n-k}{2} \), then \( \Lambda(x) = \gamma \Lambda_e(x) \) for any nonzero \( \gamma \in F \) is a nonzero polynomial of the smallest degree that satisfies \( \deg (Y(x)\Lambda(x) \mod m(x)) < (n+k)/2 \).

**Proof of Theorem 2.3** From (2.31), we have

\[
\deg (Y(x)\Lambda_e(x) \mod m(x)) < k + w_H(e),
\]

(2.38)

and (2.35) follows from \( k + w_H(e) \leq k + \frac{n-k}{2} = \frac{n+k}{2} \).

As for the converse, assume (2.36), (2.37), and \( \deg \Lambda(x) \leq t \) and consider

\[
Y(x)\Lambda(x) \mod m(x) = C(x)\Lambda(x) + E(x)\Lambda(x) \mod m(x).
\]

(2.39)
Under the stated assumptions, the degree of the left-hand side of (2.39) is smaller than $n - t$ and also

$$\deg \left( C(x)\Lambda(x) \right) < k + t \leq n - t. \quad (2.40)$$

It follows that

$$\deg \left( E(x)\Lambda(x) \mod m(x) \right) < n - t. \quad (2.41)$$

Now write

$$E(x)\Lambda(x) = Q(x)m(x) + E(x)\Lambda(x) \mod m(x) \quad (2.42)$$

according to the polynomial division theorem. But $E(x)$ (and thus $E(x)\Lambda(x)$) has at least $n - w_H(e) \geq n - t$ zeros in the set $\{\beta_0, \beta_1, \ldots, \beta_{n-1}\}$. It follows that $E(x)\Lambda(x) \mod m(x)$ has also at least $n - t$ zeros (in this set), which contradicts (2.41) unless

$$E(x)\Lambda(x) \mod m(x) = 0. \quad (2.43)$$

But any nonzero polynomial $\Lambda(x)$ that satisfies (2.43) is a multiple of the error locator polynomial $\gamma \Lambda_e(x)$

To find such $\Lambda(x) = \gamma \Lambda_e(x)$ that satisfies (2.37) turns out to be a Partial-Inverse Problem of Section 2.1. We thus arrive at the following decoding procedure:

1. Compute $Y(x) = \psi^{-1}(y)$.

2. Run any algorithm of Sections 2.3 and 2.4 with $b(x) = Y(x)$ and $d = \lceil \frac{n+k}{2} \rceil$. If $w_H(e) \leq \frac{n-k}{2}$, then the polynomial $\Lambda(x)$ returned by the algorithm equals $\Lambda_e(x)$ up to a scale factor.


Note that in Step 2, because of (2.6), coefficients $Y_\ell$ of $Y(x)$ with

$$\ell < \ell_{\min} \triangleq \begin{cases} k, & \text{if } n - k \text{ is even} \\ k + 1, & \text{if } n - k \text{ is odd} \end{cases} \quad (2.44)$$

are irrelevant for finding $\Lambda(x)$ and can be set to zero. The remaining coefficients $Y_\ell$ are syndromes since $C_\ell = 0$ and $Y_\ell = E_\ell$ for $\ell \geq \ell_{\min}$. In fact, coefficients $m_\ell$ of $m(x)$ with $\ell$ that satisfies (2.7) are also irrelevant.
for the solution $\Lambda(x)$ and can also be set to zero. This observation leads to Remark 1 of Section 2.5.5 and agrees with the observation in the end of Section 2.5.4.

Note that that in Step 3, computing the numerator of (2.29) may be viewed as continuing the PIA of Section 2.3 (line 36) with frozen $\Lambda^{(1)}(x) = \Lambda(x)$, or continuing QSA of Section 2.4.1 (line 36) with frozen $\Lambda^{(1)}(x) = \Lambda(x)$ and frozen $\Gamma^{(1)}(x)$, depending on which algorithm is used in Step 2; in any case, i.e, for arbitrary (2.20), we have a Berlekamp-Massey-like algorithm.

Remark: RSA of Section 2.4.2 offers an alternative for solving (2.37). Using RSA, (2.29) can then be carried out by $C(x) = r^{(1)}(x)/\Lambda(x)$ because $r^{(1)}(x) = Y(x)\Lambda(x) \mod m(x) = C(x)\Lambda(x)$. The resulting algorithm is then equivalent to Shiozaki’s [19] and Gao’s [20] algorithm for decoding Reed-Solomon codes.

### 2.5.4 Decoding With The Transformed Key Equation

The error-locator polynomial (2.27) can also be found by solving the transformed key equation (2.52) below, which looks like the standard (conventional) key equation that will be discussed in Section 2.9. Note, however, that (2.25) is not required in deriving (2.52).

Recall (2.20) and recall that $\deg m(x) = n$. Let

$$\overline{m}(x) \triangleq x^n m(x^{-1}) \quad (2.45)$$

and let

$$\overline{Y}(x) \triangleq x^{n-1} Y(x^{-1}). \quad (2.46)$$

Note that $\overline{m}(0) = 1$ and therefore $\overline{m}(x)$ is relatively prime to $x^\ell$ for any $\ell \geq 1$, which implies that the inverse of $\overline{m}(x)$ (mod $x^\ell$) in $F[x]/x^\ell$ exists. We will set $\ell = n - k$ and define $W(x)$ as the inverse of $\overline{m}(x)$ mod $x^{n-k}$ in this ring $F[x]/x^{n-k}$; such $W(x)$ can be pre-computed from the given $m(x)$.

Now let

$$\tilde{Y}(x) \triangleq \overline{Y}(x) W(x) = \sum_{\ell=0}^{n-k} \tilde{Y}_{\ell} x^\ell \quad (2.47)$$

and let

$$\overline{S}(x) \triangleq \sum_{\ell=0}^{n-k-1} \tilde{Y}_{\ell} x^\ell. \quad (2.48)$$
We then have the syndrome polynomial
\[ S(x) \triangleq x^{n-k-1}\overline{S}(x^{-1}) \] (2.49)
and the following theorem.

**Theorem 2.4 (Transformed Key Equation)** If \( w_H(e) \leq \frac{n-k}{2} \), then the error locator polynomial \( \Lambda_e(x) \) satisfies
\[ \deg \left( S(x)\Lambda_e(x) \mod x^{n-k} \right) < \frac{(n-k)}{2}. \] (2.50)

Conversely, for any \( y \) and \( e \in F^n \) and \( t \in \mathbb{R} \) with
\[ w_H(e) \leq t \leq \frac{(n-k)}{2}, \] (2.51)
if some nonzero \( \Lambda(x) \in F[x] \) with \( \deg \Lambda(x) \leq t \) satisfies
\[ \deg \left( S(x)\Lambda(x) \mod x^{n-k} \right) < n-k-t, \] (2.52)
then \( \Lambda(x) \) is a multiple of \( \Lambda_e(x) \).

The proof of Theorem 2.4 will be given in Section 2.10.

Clearly, to find such \( \Lambda(x) = \gamma \Lambda_e(x) \) that satisfies (2.52) is a Partial-Inverse Problem of Section 2.1. We thus have the following decoding procedure:

1. Compute \( Y(x) = \psi^{-1}(y) \) and \( S(x) \) from (2.47)–(2.49).

2. Run the **PIA** algorithm of Section 2.3 with \( S(x), x^{n-k} \), and \( d = \lceil \frac{n-k}{2} \rceil \) as the corresponding inputs. If \( w_H(e) \leq \frac{n-k}{2} \), then the polynomial \( \Lambda(x) \) returned by the algorithm equals \( \Lambda_e(x) \) up to a scale factor.


Note that in Step 2, with \( S(x) \) and \( x^{n-k} \) as the inputs, the **PIA** algorithm will use line 41 (of Section 2.3) to compute \( \kappa_1 \). The complexity of Step 2 will agree with Berlekamp-Massey algorithm, in any case of (2.20).

In the special cases where (2.20) equals (2.24), \( m(x) = x^n - x \) (when 0 is used as an evaluation point), and \( m(x) = x^n - p(x) \) with \( \deg p(x) \leq k \), we have \( W(x) = 1 \). For these cases, the transforming technique amounts to picking the upper \( n-k \) coefficients of \( Y(x) = \sum_{\ell} Y_\ell x^\ell \) to form
\[ S(x) = Y_k + Y_{k+1}x + \ldots + Y_{n-1}x^{n-k-1} \] (2.53)
and to treating \( m(x) \) as if \( m(x) = x^{n-k} \). This observation agrees with the one made in Section 2.5.3, and also leads to Remark 1 of Section 2.5.5.
2.5.5 Concluding Remarks: Berlekamp-Massey-Like Computation for any \( m(x) \)

Remark 1: in the special cases where \( m(x) = x^n - 1 \), \( m(x) = x^n - x \), and \( m(x) = x^n - p(x) \) with \( \deg p(x) \leq k \), we may form \( b(x) = Y_k + Y_{k+1}x + \ldots + Y_{n-1}x^{n-k-1} \), and feed such \( b(x) \) together with \( x^{n-k} \) and \( d = \lceil n - k \rceil / 2 \) as the corresponding inputs into PIA of Section 2.3. The PIA will find \( \Lambda_e(x) \), provided that \( w_H(e) \leq (n - k)/2 \); moreover, the PIA (with the replacement of line 36 by line 41) computes \( \Lambda_e(x) \) in the same fashion as the Berlekamp-Massey algorithm.

Remark 2: for arbitrary (2.20) (excluding these special cases), we can compute \( \Lambda_e(x) \) either by solving (2.37) as in Section 2.5.3 by directly using QSA of Section 2.4, or by solving (2.52) as in Section 2.5.4 by using PIA with line 41; the later method requires a pre-computation of (2.49), which amounts to a multiplication of two polynomials of degree less than \( n - k \), cf. (2.47) and (2.48).

In any case, we have a reverse Berlekamp-Massey-like algorithm for computing \( \Lambda_e(x) \), which works naturally and efficiently for any \( m(x) \).

2.6 Key Elements of The Proof

In this section, we discuss some key elements of the proof of the PIA proposed in Section 2.3; the actual proof will then be given in Section 2.7.

The pivotal part of PIA is line 7, which is explained by the following simple lemma. (The corresponding statement for the Berlekamp-Massey algorithm is known as the two-wrongs-make-a-right lemma, so called by J. L. Massey.)

**Lemma 2.1** Let \( m(x) \) be a polynomial over \( F \) with \( \deg m(x) \geq 1 \). For further polynomials \( b(x), \Lambda^{(1)}(x), \Lambda^{(2)}(x) \in F[x] \), let

\[
\begin{align*}
    r^{(1)}(x) & \triangleq b(x)\Lambda^{(1)}(x) \mod m(x), \\
    r^{(2)}(x) & \triangleq b(x)\Lambda^{(2)}(x) \mod m(x),
\end{align*}
\]

\( d_1 \triangleq \deg r^{(1)}(x) \), \( \kappa_1 \triangleq \lcf r^{(1)}(x) \), \( d_2 \triangleq \deg r^{(2)}(x) \), \( \kappa_2 \triangleq \lcf r^{(2)}(x) \), and assume \( d_1 \geq d_2 \geq 0 \). Then

\[
\Lambda(x) \triangleq \kappa_2\Lambda^{(1)}(x) - \kappa_1 x^{d_1 - d_2} \Lambda^{(2)}(x)
\]

satisfies

\[
\deg \left( b(x)\Lambda(x) \mod m(x) \right) < d_1.
\]
**Proof:** From (2.56), we obtain
\[ r(x) \triangleq b(x)\Lambda(x) \mod m(x) \quad (2.58) \]
\[ = \kappa_2 r^{(1)}(x) - \kappa_1 x^{d_1-d_2} r^{(2)}(x) \quad (2.59) \]
by the natural ring homomorphism \( F[x] \to F[x]/m(x) \). It is then obvious from (2.59) that \( \deg r(x) < \deg r^{(1)}(x) = d_1 \). \( \square \)

The following lemma is the counterpart to Theorem 1 of \cite{3}.

**Lemma 2.2 (Degree Change Lemma)** For fixed nonzero \( b(x) \) and \( m(x) \in F[x] \) with \( \deg b(x) < \deg m(x) \), let \( \Lambda(x) \) be a minimal partial inverse of \( b(x) \) and let
\[ r(x) \triangleq b(x)\Lambda(x) \mod m(x). \quad (2.60) \]
Then any nonzero polynomial \( \Lambda^{(1)}(x) \in F[x] \) such that
\[ \deg \left( b(x)\Lambda^{(1)}(x) \mod m(x) \right) < \deg r(x) \quad (2.61) \]
satisfies
\[ \deg \Lambda^{(1)}(x) \geq \deg m(x) - \deg r(x). \quad (2.62) \]

The proof is given below.

**Corollary 2.3** Assume everything as in Lemma 2.2 including (2.61). If (2.62) is satisfied with equality, then \( \Lambda^{(1)}(x) \) is also a minimal partial inverse of \( b(x) \).

**Proof of Lemma 2.2:** Assume that \( \Lambda^{(1)}(x) \) is a nonzero polynomial that satisfies (2.61), i.e., the degree of
\[ r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \mod m(x) \quad (2.63) \]
satisfies
\[ \deg r^{(1)}(x) < \deg r(x). \quad (2.64) \]
Multiplying (2.60) by \( \Lambda^{(1)}(x) \) and (2.63) by \( \Lambda(x) \) yields
\[ \Lambda^{(1)}(x)r(x) \equiv \Lambda(x)r^{(1)}(x) \mod m(x). \quad (2.65) \]
Note that
\[ \deg \Lambda^{(1)}(x) \geq \deg \Lambda(x) \quad (2.66) \]
because $\Lambda(x)$ is a minimal partial inverse and because of (2.61) (cf. the
definition of minimal partial inverse). We thus have
\[
\deg \Lambda^{(1)}(x) + \deg r(x) > \deg \Lambda(x) + \deg r^{(1)}(x).
\]
(2.67)
If we assume (contrary to (2.62))
\[
\deg \Lambda^{(1)}(x) < \deg m(x) - \deg r(x),
\]
(2.68)
then (2.65) reduces to
\[
\Lambda^{(1)}(x)r(x) = \Lambda(x)r^{(1)}(x),
\]
(2.69)
which is a contradiction to (2.67). □

2.7 Proof of The Algorithm

In this section, we first prove the correctness of the PIA proposed in
Section 2.3. We then prove Proposition 2.7, which implies the correctness
of QSA of Section 2.4.1.

2.7.1 Correctness of the Proposed Algorithm

To this end, we restate the algorithm with added assertions as follows.

Proposed Algorithm Restated:
1   if $\deg b(x) < d$ begin
2      return $\Lambda(x) := 1$
3   end
4   $\Lambda^{(1)}(x) := 0$, $d_1 := \deg m(x)$, $\kappa_1 := \lcf m(x)$
5   $\Lambda^{(2)}(x) := 1$, $d_2 := \deg b(x)$, $\kappa_2 := \lcf b(x)$
6   loop begin
7      repeat
8         $\Lambda^{(1)}(x) := \kappa_2 \Lambda^{(1)}(x) - \kappa_1 x^{d_1 - d_2} \Lambda^{(2)}(x)$
    Assertions:
    $d_1 > d_2 \geq d$ (A.1)
    $\deg \Lambda^{(2)}(x) = \deg m(x) - d_1$ (A.2)
    $> \deg \Lambda^{(1)}(x)$ (A.3)
    $\Lambda^{(2)}(x)$ is a minimal partial inverse (A.4)
### Assertions:

- \[ \deg(b(x)\Lambda^{(1)}(x) \mod m(x)) < d_1 \quad (A.5) \]
- \[ \deg \Lambda^{(1)}(x) = \deg m(x) - d_2 \quad (A.6) \]
- \[ > \deg \Lambda^{(2)}(x) \quad (A.7) \]

9. \[ d_1 := \deg(b(x)\Lambda^{(1)}(x) \mod m(x)) \]

10. if \( d_1 < d \) begin

   **Assertion:**
   \[ \Lambda^{(1)}(x) \text{ is a min. partial inverse} \quad (A.8) \]

11. return \( \Lambda(x) := \Lambda^{(1)}(x) \)

12. end

13. \[ \kappa_1 := \lcf(b(x)\Lambda^{(1)}(x) \mod m(x)) \]

14. until \( d_1 < d_2 \) begin

   **Assertion:**
   \[ \Lambda^{(1)}(x) \text{ is a minimal partial inverse} \quad (A.9) \]

15. \((\Lambda^{(1)}(x), \Lambda^{(2)}(x)) := (\Lambda^{(2)}(x), \Lambda^{(1)}(x))\)

16. \((d_1, d_2) := (d_2, d_1)\)

17. \((\kappa_1, \kappa_2) := (\kappa_2, \kappa_1)\)

end

Note the added inner repeat loop (lines 7–14), which does not change the algorithm but helps to state its proof.

Throughout the algorithm (except at the very beginning, before the first execution of lines 9 and 13), \( d_1, d_2, \kappa_1, \) and \( \kappa_2 \) are defined as in Lemma 2.1, i.e., \( d_1 = \deg r^{(1)}(x) \), \( \kappa_1 = \lcf r^{(1)}(x) \), \( d_2 = \deg r^{(2)}(x) \), and \( \kappa_2 = \lcf r^{(2)}(x) \) for \( r^{(1)}(x) \) and \( r^{(2)}(x) \) as in (2.54) and (2.55).

Assertions (A.1)–(A.4) are easily verified, both from the initialization and from (A.6), (A.7), and (A.9).

As for (A.5), after the very first execution of line 8 we still have \( d_1 = \deg m(x) \) (from line 4), which makes (A.5) obvious. For all later executions of line 8 (A.5) follows from Lemma 2.1.

As for (A.6) and (A.7), we note that line 8 changes the degree of \( \Lambda^{(1)}(x) \) as follows:

- Upon entering the repeat loop, line 8 increases the degree of \( \Lambda^{(1)} \) to

  \[ \deg \Lambda^{(2)}(x) + d_1 - d_2 = \deg m(x) - d_2 \]
  \[ > \deg \Lambda^{(2)}(x), \quad (2.70) \]

  which follows from (A.1)–(A.3).
2.7 Proof of The Algorithm

• Subsequent executions of line 8 without leaving the repeat loop (i.e., without executing lines 15–17) do not change the degree of \( \Lambda^{(1)}(x) \). (This follows from the fact that \( d_1 \) is smaller than in the first execution while \( \Lambda^{(2)}(x) \), \( d_2 \), and \( \kappa_2 \neq 0 \) remain unchanged.)

Assertion (A.9) follows from the Corollary to Lemma 2.2 (with \( \Lambda(x) = \Lambda^{(2)}(x) \) and \( \deg r(x) = d_2 \), which applies because \( d_1 < d_2 \) and (A.6). Because of (A.1), the same argument applies also to (A.8).

Finally, (A.1) and (A.6) imply that the polynomial \( \Lambda(x) \) returned by the algorithm satisfies

\[
\deg \Lambda(x) \leq \deg m(x) - d. \tag{2.72}
\]

2.7.2 Proving Propositions 2.5 and 2.6

Proposition 2.5 should be clear from the proof of Section 2.7.1 for PIA. In fact, the proof (of PIA) is also a constructive proof for Proposition 2.6: the proposition follows from that the restated algorithm of Section 2.7.1 (with \( d = 1 \)) starts with the minimal partial inverse \( \Lambda^{(2)}(x) := 1 \) (line 5) of the smallest degree, and then finds, progressively and recursively, right the next (by Lemma 2.2) minimal partial inverse \( \Lambda^{(1)}(x) \) of larger degree between lines 14–15 (see Assertion (A.9)).

2.7.3 Proof of Proposition 2.7

We begin with the following lemma.

Lemma 2.3 Let \( m(x) \in F[x] \) with \( \deg m(x) \geq 1 \). For further polynomials \( b(x), \Lambda^{(1)}(x), \Lambda^{(2)}(x) \in F[x] \), assume that

\[
\begin{align*}
    b(x)\Lambda^{(1)}(x) \mod m(x) &= b(x)\Lambda^{(1)}(x) + m(x)\Gamma^{(1)}(x) \tag{2.73} \\
    b(x)\Lambda^{(2)}(x) \mod m(x) &= b(x)\Lambda^{(2)}(x) + m(x)\Gamma^{(2)}(x) \tag{2.74}
\end{align*}
\]

hold respectively for some \( \Gamma^{(1)}(x) \) and \( \Gamma^{(2)}(x) \). Then

\[
\begin{align*}
    \Lambda(x) &\triangleq \kappa_2\Lambda^{(1)}(x) - \kappa_1x^{d_1-d_2}\Lambda^{(2)}(x) \tag{2.75} \\
    \Gamma(x) &\triangleq \kappa_2\Gamma^{(1)}(x) - \kappa_1x^{d_1-d_2}\Gamma^{(2)}(x) \tag{2.76}
\end{align*}
\]

with \( \kappa_1, \kappa_2 \in F \) and \( d_1 \geq d_2 \geq 0 \in \mathbb{Z} \) together satisfy

\[
b(x)\Lambda(x) \mod m(x) = b(x)\Lambda(x) + m(x)\Gamma(x) \tag{2.77}
\]
The lemma can be proved by a straightforward calculation.

We now begin to prove Proposition 2.7. Our proof is again based on the restated algorithm of Section 2.7.1 and accordingly, we add the additional initialization

\[ \Gamma^{(1)}(x) := 1, \quad \Gamma^{(2)}(x) := 0 \]

into line 5 of the restated algorithm. Correspondingly,

\[ \Gamma^{(1)}(x) := \kappa_2 \Gamma^{(1)}(x) - \kappa_1 x^{d_1 - d_2} \Gamma^{(2)}(x) \]

is added also into line 8 and the swap

\[ (\Gamma^{(1)}(x), \Gamma^{(2)}(x)) := (\Gamma^{(2)}(x), \Gamma^{(1)}(x)) \]

is added into line 15.

We first observe from the initialization (line 5) that the condition

\[ b(x) \Lambda^{(2)}(x) \mod m(x) = b(x) \Lambda^{(2)}(x) + m(x) \Gamma^{(2)}(x) \quad (2.78) \]

hold between lines 6–7. Then after the very first execution of line 8, we obtain

\[ \Lambda^{(1)}(x) = -\kappa_1 x^{d_1 - d_2} \quad \text{and} \quad \Gamma^{(1)}(x) = \kappa_2, \quad (2.79) \]

which clearly satisfy

\[ b(x) \Lambda^{(1)}(x) \mod m(x) = b(x) \Lambda^{(1)}(x) + m(x) \Gamma^{(1)}(x). \quad (2.80) \]

Therefore (2.78) and (2.80) hold between lines 8–9.

For all the later execution of line 8 and iteration of the algorithm, we apply Lemma 2.3 and thus, we always have (2.80) between lines 8–9. Together with \((\Lambda, 5)\), we prove that \(b(x) \Lambda^{(1)}(x) \mod m(x)\) of line 9 (of the restated algorithm) indeed amounts to

\[ b(x) \Lambda^{(1)}(x) \mod m(x) = \sum_{\ell=0}^{\tau} b_{d_1 - \tau} \Lambda^{(1)}_{\ell} + \sum_{\ell=0}^{\nu} m_{d_1 - \tau} \Gamma^{(1)}_{\ell} \quad (2.81) \]

with \(\tau \triangleq \deg \Lambda^{(1)}(x)\) and \(\nu \triangleq \deg \Gamma^{(1)}(x)\), and where both \(b_{\ell} \triangleq 0\) and \(m_{\ell} \triangleq 0\) for \(\ell < 0\).
2.8 Application to Padé Approximation

The algorithms of Sections 2.3 and 2.4 can be applied to compute Padé approximants.

Padé Approximation \[12\]: Let \( F \) be some field, let \( n \) be a positive integer, and let \( B(x) = b_0 + b_1 x + b_2 x^2 + \ldots \) be a power series over \( F \). A \((u,v)\) Padé approximant to \( B(x) \), with \( u + v = n - 1 \), is a rational function \( r(x)/\Lambda(x) \) such that \( r(x) \) and \( \Lambda(x) \) are a pair of polynomials of the smallest degree that satisfy the following conditions:\(^2\)

\[
\begin{align*}
\deg r(x) & \leq u, \quad (2.82) \\
\deg \Lambda(x) & \leq v, \quad (2.83)
\end{align*}
\]

and

\[
B(x) \equiv \frac{r(x)}{\Lambda(x)} \mod x^n. \quad (2.84)
\]

**Proposition 2.10** If the \((u,v)\) Padé approximant \( r(x)/\Lambda(x) \) to \( B(x) \) exists, where \( u + v = n - 1 \), then finding such \( \Lambda(x) \) and \( r(x) \) (of the smallest degree) that satisfy \((2.82)–(2.84)\) is a Partial-Inverse Problem of Section 2.1 for \( b(x) = B(x) \mod x^n \), \( m(x) = x^n \), and \( d = u + 1 \).

**Proof:** Note first that \((2.84)\) is equivalent to \( b(x)\Lambda(x) \equiv r(x) \mod x^n \) for \( b(x) = B(x) \mod x^n \). We then note that because of \((2.4)\), the resulting \( \Lambda(x) \) satisfies \( \deg \Lambda(x) \leq n - u - 1 = v \), which is \((2.83)\), and we have \( r(x) = b(x)\Lambda(x) \mod x^n \) with \( \deg r(x) \leq d - 1 = u \), which is \((2.82)\). □

The denominator \( \Lambda(x) \) of a \((u,v)\) Padé approximant turns out to be a minimal partial inverse of \( b(x) \), up to a scale factor.

2.9 Application to Solving Standard Key Equation

Reed-Solomon codes are conventionally \[2,3,9–11,21\] defined as a class of cyclic codes with \((2.24)\), i.e., the codes as in \((2.22)\) but satisfy the

\(^2\)It may be required that \( \Lambda(0) = 1 \), up to the definition of rational function. It may also be required that \( r(x) \) and \( \Lambda(x) \) are relatively prime depending on how one defines Padé approximant; here we prefer to include such a condition in addition to \((2.82)–(2.84)\). Since we require \( r(x) \) and \( \Lambda(x) \) to be relatively prime, such \((u,v)\) Padé approximant \( r(x)/\Lambda(x) \) may not always exist.
condition (2.23). At the heart of decoding such codes is on solving a key equation. Note that there are different forms of key equation (e.g., [2,3,9–11,21]), but under (2.23), they can be reduced to the form as in (2.85) below together with (2.86) and (2.87), as noted in [21]. We will call (2.85)–(2.87) as the standard key equation for decoding Reed-Solomon codes.

A Reed-Solomon code, or say a generalized Reed-Solomon code, may also be defined under the more relaxed condition (2.25) than (2.23). It can be shown that such codes can also be decoded based on solving equations of the same form as (2.85)–(2.87), cf. [11] or Section 2.10.2. In the following, we show that solving the standard key equation (2.85)–(2.87) is actually a Partial-Inverse Problem.

The standard key equation (SKE) for decoding Reed-Solomon codes [2,9–11,21] is of the form

\[ S_c(x)\Lambda(x) \equiv r(x) \mod x^{n-k}, \tag{2.85} \]

where \( n \) and \( k \) are the blocklength and the dimension of the code, respectively, and where \( S_c(x) \) is a (given) syndrome polynomial with \( \deg S_c(x) < n - k \) associated with an error pattern \( e \). The desired solution (under the assumption \( w_H(e) \leq (n-k)/2 \)) is a pair \( r(x) \) and \( \Lambda(x) \neq 0 \) such that

\[ \deg r(x) < \deg \Lambda(x) \leq (n-k)/2 \tag{2.86} \]

and

\[ \gcd (r(x), \Lambda(x)) = 1. \tag{2.87} \]

In the literature, e.g., [2,3,9,11], \( \Lambda(x) \) is an error locator polynomial that is defined as

\[ \Lambda(x) \triangleq \prod_{\ell \in \{0, \ldots, n-1\}, e\ell \neq 0} (1 - \beta_{e\ell}x) \tag{2.88} \]

instead of (2.27).\(^3\) Moreover, when deriving (2.85)–(2.87), it is usually assumed either implicitly or explicitly that 0 is not used as an evaluation point in defining Reed-Solomon codes.

**SKE Problem**: For the given syndrome polynomial \( S_c(x) \neq 0 \) associated with \( e \) of \( w_H(e) \leq (n-k)/2 \), find such a pair \( \Lambda(x) \) and \( r(x) \) that satisfy (2.85)–(2.87).

\(^3\)In fact, (2.88) and (2.27) are related by \( \Lambda(x) = x^{w_H(e)}\Lambda_e(x^{-1}) \).
Proposition 2.11  Any \( \Lambda(x) \) that satisfies (2.85)–(2.87) for some \( r(x) \), if exists, is a minimal partial inverse of \( b(x) \triangleq S_c(x) \) (for fixed \( m(x) = x^{n-k} \)), and thus it is unique.

Proof: Assume that such \( \Lambda(x) \) is not a minimal partial inverse, i.e., there exists \( \Lambda^{(1)}(x) \) with \( \deg \Lambda^{(1)}(x) < \deg \Lambda(x) \) such that the resulting

\[
r^{(1)}(x) \triangleq b(x)\Lambda^{(1)}(x) \mod x^{n-k}
\]

satisfies \( \deg r^{(1)}(x) \leq \deg r(x) \). Multiplying (2.85) by \( \Lambda^{(1)}(x) \) and (2.89) by \( \Lambda(x) \) then yields

\[
r^{(1)}(x)\Lambda(x) = r(x)\Lambda^{(1)}(x)
\]

because of (2.86). It follows that \( \Lambda(x) \) divides \( \Lambda^{(1)}(x) \) since \( \Lambda(x) \) and \( r(x) \) are relatively prime. But we have \( \deg \Lambda(x) > \deg \Lambda^{(1)}(x) \), which leads to a contradiction. Therefore, \( \Lambda(x) \) must be a minimal partial inverse,

Uniqueness follows from Proposition 2.2. □

We thus have the corollary.

Corollary 2.4  The SKE problem is a Partial-Inverse Problem of the setting \( b(x) = S_c(x) \), \( m(x) = x^{n-k} \), and \( d = \lceil (n-k)/2 \rceil \).

Proof: With \( d = \lceil (n-k)/2 \rceil \) and because of (2.4), the returned \( \Lambda(x) \) satisfies \( \deg \Lambda(x) \leq (n-k)/2 \). On the other hand, the resulting \( r(x) = b(x)\Lambda(x) \mod x^{n-k} \) trivially satisfies \( \deg r(x) < (n-k)/2 \). The \( \Lambda(x) \) is indeed the one that we are looking for, because, by Lemma 2.2, any minimal partial inverse \( \tilde{\Lambda}(x) \) such that \( \deg (b(x)\tilde{\Lambda}(x) \mod x^{n-k}) < \deg r(x) \) satisfies \( \deg \tilde{\Lambda}(x) \geq n-k - \deg r(x) > (n-k)/2 \).

Finally, we remark that in Section 2.10, we describe another way to derive the standard key equation.

2.10 Proving The Transformed Key Equation and Deriving Standard Key Equation

In this section, we give a proof of Theorem 2.4. The proving technique also leads to a derivation of the standard key equation of Section 2.9.
2.10.1 Deriving and Proving Theorem 2.4

Follow the setup of Section 2.5. Let $\beta_0, \ldots, \beta_{n-1}$ be $n$ different elements of some finite field $F$, and let $C$ be an $(n,k)$ Reed-Solomon code defined as in (2.22) with $\psi$ as in (2.21). Note that the condition (2.25) is not required.

Let $y = c + e \in F^n$ be a received word where $c \in C$ and where $e$ is an error pattern. Moreover, let $Y(x) \triangleq \psi^{-1}(y) = C(x) + E(x)$ where $C(x) \triangleq \psi^{-1}(c)$ with $\deg C(x) < k$, and $E(x) \triangleq \psi^{-1}(e)$ with $\deg E(x) < \deg m(x) = n$.

We start from noting that there exist $\Lambda(x) \not= 0$ and $r(x) \triangleq Y(x)\Lambda(x) \mod m(x)$ (2.91) such that $\deg r(x) < n - t$ and $\deg \Lambda(x) \leq t$ for any $0 \leq t \leq \deg m(x)$, cf. Theorem 2.2. In the following, we assume $w_H(e) \leq t \leq (n - k)/2$, since here we are interested in proving Theorem 2.4.

Now rewrite (2.91) as follows

$$Y(x)\Lambda(x) - r(x) = q(x)m(x)$$ (2.92)

for some $q(x)$ with $\deg q(x) < \deg \Lambda(x) \leq t$. Furthermore, define the polynomials $\overline{\Lambda}(x), \overline{Y}(x), \overline{q}(x), \overline{m}(x)$, and $\overline{r}(x)$ by

$$\overline{\Lambda}(x) \triangleq x^t\Lambda(x^{-1})$$ (2.93)
$$\overline{Y}(x) \triangleq x^{n-1}Y(x^{-1})$$ (2.94)
$$\overline{q}(x) \triangleq x^{t-1}q(x^{-1})$$ (2.95)
$$\overline{m}(x) \triangleq x^nm(x^{-1})$$ (2.96)
$$\overline{r}(x) \triangleq x^{n-t-1}r(x^{-1})$$ (2.97)

and reverse both sides of (2.92) by

$$x^{n+\tau-1}(Y(x^{-1})\Lambda(x^{-1}) - r(x^{-1})) = x^{n+\tau-1}q(x^{-1})m(x^{-1})$$ (2.98)

where

$$\tau \triangleq (n - k)/2.$$ (2.99)

We then obtain

$$\overline{Y}(x)x^{\tau - t}\overline{\Lambda}(x) - x^{\tau + t}\overline{r}(x) = x^{\tau - t}\overline{q}(x)\overline{m}(x)$$ (2.100)
2.10 Proving The Transformed Key Equation and Deriving Standard Key Equation

and then

\[ \overline{Y}(x)x^{2(\tau-t)\overline{\Lambda}(x)} \equiv x^{2(\tau-t)\overline{q}(x)\overline{m}(x)} \mod x^{2\tau} \quad (2.101) \]

and thus

\[ \overline{Y}(x)x^{n-k-2t\overline{\Lambda}(x)} \equiv x^{n-k-2t\overline{q}(x)\overline{m}(x)} \mod x^{n-k} \quad (2.102) \]

Let \( W(x) \) be the inverse of \( m(x) \mod x^{n-k} \) in \( F[x]/x^{n-k} \); such \( W(x) \) exists since \( m(0) = 1 \), which implies that \( m(x) \) is relatively prime to \( x^{n-k} \). We then have

\[ \overline{Y}(x)x^{n-k-2t\overline{\Lambda}(x)}W(x) \equiv x^{n-k-2t\overline{q}(x)} \mod x^{n-k} \quad (2.103) \]

Further, let

\[ \tilde{Y}(x) \triangleq \overline{Y}(x)W(x) = \sum_{\ell=0}^{\tilde{Y}_\ell} x^\ell \quad (2.104) \]

and

\[ \overline{S}(x) \triangleq \sum_{\ell=0}^{n-k-1} \tilde{Y}_\ell x^\ell \quad (2.105) \]

which is (2.48). Congruence (2.103) can finally be written as

\[ x^{n-k-2t\overline{\Lambda}(x)}\overline{S}(x) \equiv x^{n-k-2t\overline{q}(x)} \mod x^{n-k} \quad (2.106) \]

Note that the condition \( \deg q(x) < \deg \Lambda(x) \leq t \) implies both that (2.93) satisfies \( \deg \overline{\Lambda}(x) \leq t \) and that (2.95) satisfies \( \deg \overline{q}(x) < t \). It is then clear that

\[ \deg \left( x^{n-k-2t\overline{\Lambda}(x)} \right) \leq n - k - t \quad (2.107) \]

and

\[ \deg \left( x^{n-k-2t\overline{q}(x)} \right) < n - k - t \quad (2.108) \]

Now write (2.106) as

\[ x^{n-k-2t\overline{\Lambda}(x)}\overline{S}(x) - x^{n-k-2t\overline{q}(x)} = \overline{p}(x)x^{n-k} \quad (2.109) \]

for some \( \overline{p}(x) \) with \( \deg \overline{p}(x) < n - k - t \). Furthermore, define the polynomials

\[ p(x) \triangleq x^{n-k-t-1}\overline{p}(x^{-1}) \quad (2.110) \]

and

\[ S(x) = x^{n-k-1}\overline{S}(x^{-1}) \quad (2.111) \]
By substituting every $x$ of (2.109) by $x^{-1}$ and multiplying both sides of (2.109) by $x^{2(n-k)-t-1}$, we then obtain

$$\Lambda(x)S(x) - x^{n-k}q(x) = p(x) \quad (2.112)$$

and thus the polynomial $\Lambda(x)$ with $\deg \Lambda(x) \leq t$ satisfies

$$p(x) = \Lambda(x)S(x) \mod x^{n-k} \quad (2.113)$$

for some $p(x)$ with $\deg p(x) < n - k - t$.

Theorem 2.4 then follows from Theorem 2.3 together with the above derivation.

Remark: By Theorem 2.3, if $w_H(e) \leq (n - k)/2$, then $\Lambda(x) = \Lambda_e(x)$ is a polynomial of the smallest degree that satisfies (2.92) for some $q(x)$ with $\deg q(x) < \deg \Lambda_e(x)$. In fact, we have $q(x) = E(x)\Lambda_e(x)/m(x)$; this can be easily seen from (2.28) and from that the coefficients $Y_\ell$ of $Y(x)$ equals $E_\ell$ for $\ell \geq \ell_{\text{min}}$ of (2.44). We can further show that $\gcd(q(x), \Lambda_e(x)) = 1$. Indeed, $q(\beta_\ell) \neq 0$ whenever $\Lambda_e(\beta_\ell) = 0$:

$$\Lambda_e(\beta_\ell) = 0 \iff e_\ell \neq 0 \iff E(\beta_\ell) \neq 0; \quad (2.114)$$

on the other hand, $m_S(\beta_\ell) \neq 0$ where $m_S(x) \overset{\Delta}{=} \Lambda_e(x)/m(x)$. Therefore, $q(\beta_\ell) = E(\beta_\ell)m_S(\beta_\ell) \neq 0$.

### 2.10.2 A Way to Derive the Standard Key Equation

The derivation of the previous section up to (2.106) also leads to a derivation of the standard key equation of Section 2.9.

Assume (2.25), i.e., $\beta_\ell \neq 0$ for all $\ell \in \{0, \ldots, n - 1\}$, and assume $w_H(e) \leq (n - k)/2$. Then, define the polynomials

$$\overline{\Lambda}_e(x) \overset{\Delta}{=} x^{w_H(e)}\Lambda_e(x^{-1}) \quad (2.115)$$

and

$$\overline{q}_e(x) \overset{\Delta}{=} x^{w_H(e)-1}q_e(x^{-1}) \quad (2.116)$$

where

$$q_e(x) \overset{\Delta}{=} E(x)\Lambda_e(x)/m(x) \quad (2.117)$$

satisfies $\deg q_e(x) < \deg \Lambda_e(x) \leq (n - k)/2$.

Note that $\deg \overline{\Lambda}_e(x) = \deg \Lambda_e(x)$ due to (2.25) and that $\deg \overline{q}_e(x) < \deg \overline{\Lambda}_e(x)$. Note also that

$$\gcd(q_e(x), \Lambda_e(x)) = 1 \quad (2.118)$$
as discussed in the end of the previous section. Since $\overline{\Lambda}_e(x)$ is not a multiple of $x$, it then follows that $\gcd\left(\overline{q}_e(x), \overline{\Lambda}_e(x)\right) = 1$ as well.

Now we are going to apply (2.106). With $t = w_H(e)$, we have

$$x^{n-k-2w_H(e)}\overline{\Lambda}_e(x)\overline{S}(x) \equiv x^{n-k-2w_H(e)}\overline{q}_e(x) \mod x^{n-k} \quad (2.119)$$

from (2.106), which in turn implies that

$$\overline{\Lambda}_e(x)\overline{S}(x) \equiv \overline{q}_e(x) \mod x^{n-k}. \quad (2.120)$$

The following theorem is then obvious.

**Theorem 2.5 (Standard Key Equation)** Let $C$ be an $(n,k)$ Reed-Solomon code defined as in (2.22) and assume (2.25). For given (2.105) (same as (2.48)), if $w_H(e) \leq (n-k)/2$, then the polynomials

$$\overline{\Lambda}_e(x) \triangleq x^{w_H(e)}\Lambda_e(x^{-1}) \quad (2.121)$$

and $\overline{q}_e(x) \triangleq \overline{S}(x)\overline{\Lambda}_e(x) \mod x^{n-k}$ together satisfy the following conditions

$$\overline{\Lambda}_e(x)\overline{S}(x) \equiv \overline{q}_e(x) \mod x^{n-k} \quad (2.122)$$

and

$$\deg \overline{q}_e(x) < \deg \overline{\Lambda}_e(x) \leq \frac{n-k}{2} \quad (2.123)$$

and

$$\gcd\left(\overline{q}_e(x), \overline{\Lambda}_e(x)\right) = 1. \quad (2.124)$$

The conditions (2.122)–(2.124) exactly agree with (2.85)–(2.87). Finally, we conclude that finding such pair $\overline{\Lambda}_e(x)$ and $\overline{q}_e(x)$ that satisfy (2.122)–(2.124) is a Partial-Inverse Problem (with $\overline{S}(x)$ and $x^{n-k}$ as the corresponding inputs) of Section 2.1 for $d = \lceil (n-k)/2 \rceil$, cf. Corollary 2.4 of Section 2.9.

### 2.11 Constrained Partial-Inverse Problem

In this section, we discuss the following generalization of the Partial-Inverse Problem.

**Constrained Partial-Inverse (CPI) Problem:** Let $b(x)$ and $m(x)$ be nonzero polynomials over some field $F$, with $\deg b(x) < \deg m(x)$. For
a given nonzero $\Lambda_c(x) \in F[x]$, find a nonzero polynomial $\Lambda(x) \in F[x]$ of the smallest degree such that $\Lambda(x)$ is a multiple of $\Lambda_c(x)$ and satisfies
\[
\deg\left( b(x)\Lambda(x) \bmod m(x) \right) < d
\]
for given $d \in \mathbb{Z}$ with $1 \leq d \leq \deg m(x)$.

We now propose two different algorithmic solutions to this problem, both of which apply to the joint errors-and-erasures decoding of Reed-Solomon codes that will be discussed in Section 2.12.

### Solution I

The CPI Problem can be reduced to the (unconstrained) Partial-Inverse Problem of Section 2.1 as follows. Assume that $\Lambda(x)$ is the solution of the CPI Problem. We then have $\Lambda(x) = \Lambda_c(x)\Lambda_v(x)$ for some $\Lambda_v(x) \neq 0$ such that $\Lambda_v(x)$ has degree as small as possible, and that
\[
\deg\left( b(x)\Lambda_c(x)\Lambda_v(x) \bmod m(x) \right) < d.
\]
With
\[
\tilde{b}(x) \triangleq b(x)\Lambda_c(x) \bmod m(x),
\]
(2.126) can then be rewritten as
\[
\deg\left( \tilde{b}(x)\Lambda_v(x) \bmod m(x) \right) < d.
\]

The CPI problem can thus be solved by the two steps:

- Run the PIA of Section 2.3 with $\tilde{b}(x)$ as the input. The returned polynomial will equal $\Lambda_v(x)$ up to a scale factor.
- We then obtain $\Lambda(x) = \Lambda_c(x)\Lambda_v(x)$.

### Solution II

The CPI problem can also be solved by the IC-PIA below, which directly returns the desired $\Lambda(x) = \gamma\Lambda_c(x)\Lambda_v(x)$ for some nonzero $\gamma \in F$. The IC-PIA is simply the PIA of Section 2.3 with $\Lambda_c(x)$ and the two quantities
\[
d_c \triangleq \deg(b(x)\Lambda_c(x) \bmod m(x))
\]
and
\[
\kappa_c \triangleq \operatorname{lcf}(b(x)\Lambda_c(x) \bmod m(x)).
\]
as the initialization of line 5.
IC-PIA Algorithm:
**Input:** $b(x)$, $m(x)$, and $\Lambda_c(x)$ as in the problem statement.
**Output:** $\Lambda(x)$ as in the problem statement.

1. if $d_c < d$
2. \hspace{0.5cm} return $\Lambda(x) := \Lambda_c(x)$
3. end
4. \hspace{0.5cm} $\Lambda^{(1)}(x) := 0$, $d_1 := \deg m(x)$, $\kappa_1 := \text{lcf} m(x)$
5. \hspace{0.5cm} $\Lambda^{(2)}(x) := \Lambda_c(x)$, $d_2 := d_c$, $\kappa_2 := \kappa_c$
6. loop begin
7. \hspace{1cm} $\Lambda^{(1)}(x) := \kappa_2 \Lambda^{(1)}(x) - \kappa_1 x^{d_1 - d_2} \Lambda^{(2)}(x)$
8. \hspace{0.5cm} $d_1 := \deg (b(x) \Lambda^{(1)}(x) \mod m(x))$
9. \hspace{0.5cm} if $d_1 < d$
10. \hspace{1cm} return $\Lambda(x) := \Lambda^{(1)}(x)$
11. \hspace{0.5cm} end
12. \hspace{0.5cm} $\kappa_1 := \text{lcf} (b(x) \Lambda^{(1)}(x) \mod m(x))$
13. if $d_1 < d_2$
14. \hspace{0.5cm} $(\Lambda^{(1)}(x), \Lambda^{(2)}(x)) := (\Lambda^{(2)}(x), \Lambda^{(1)}(x))$
15. \hspace{0.5cm} $(d_1, d_2) := (d_2, d_1)$
16. \hspace{0.5cm} $(\kappa_1, \kappa_2) := (\kappa_2, \kappa_1)$
17. \hspace{0.5cm} end
18. \hspace{0.5cm} end

Note that the computation of (2.129) and (2.130) is of no additional cost since it is the same as in lines 8 and 12.

The correctness of IC-PIA for solving the CPI problem can be easily verified from the observation: in every execution of lines 7, $\Lambda^{(1)}(x)$ of IC-PIA is exactly the multiple of the correspond $\Lambda^{(1)}(x)$ of PIA (with $\tilde{b}(x)$ as input) by $\Lambda_c(x)$.

### 2.12 Joint Errors-and-Erasures Decoding of Reed-Solomon Codes

For completeness of Section 2.5, in this section, we consider the problem of joint errors-and-erasures decoding of Reed-Solomon codes. As we will see shortly, the problem can also be reduced rather directly to the Partial-Inverse Problem of Section 2.1 or to the CPI problem of Section 2.11.
Let $C$ be an $(n, k)$ Reed-Solomon code over $F$, but without requiring (2.23) and (2.24), as in Section 2.5. Let $y = (y_0, y_1, \ldots, y_{n-1}) = c + e$ be a received word where $c \in C$ and where $e = (e_0, e_1, \ldots, e_{n-1})$ is an error pattern. Moreover, let $C(x) \triangleq \psi^{-1}(c)$ and $E(x) \triangleq \psi^{-1}(e)$ with $\psi$ as in (2.21). We then have $Y(x) \triangleq \psi^{-1}(y) = C(x) + E(x)$.

Denote by $S_u \triangleq \{\ell : e_{\ell} \neq 0, 0 \leq \ell \leq n - 1\}$ the set indexing the error positions, and by $S_v$ the set indexing the erasure locations. We will assume that $|S_u \cup S_v| = |S_u| + |S_v| \leq n - k$. For any $e \in F^n$, we define the error locator polynomial

$$\Lambda_u(x) \triangleq \prod_{\ell \in S_u} (x - \beta_{\ell})$$

and the erasure locator polynomial $\Lambda_v(x) \triangleq \prod_{\ell \in S_v} (x - \beta_{\ell})$. If $S_u$ is empty, $\Lambda_u(x) \triangleq 1$, so is for $\Lambda_v(x)$. Now, recall from Section 2.5 that $m(x) \triangleq \prod_{\ell=0}^{n-1} (x - \beta_{\ell})$.

**Theorem 2.6 (Basic Error-Erasure Correction Bound)** If

$$2 \deg \Lambda_u(x) + \deg \Lambda_v \leq n - k,$$

then (2.131) can be computed from $Y(x)$ and $m(x)$.

The theorem is a well known fact, which can also be seen from the following development, where we are going to present three approaches for the joint errors-and-erasures decoding of Reed-Solomon codes.

**Approach-I**

Let

$$\hat{Y}(x) \triangleq \Lambda_v(x)Y(x) \mod m(x)$$

$$= \hat{C}(x) + \hat{E}(x),$$

where $\hat{C}(x) \triangleq \Lambda_v(x)C(x)$ has

$$\deg \hat{C}(x) < \hat{k} \triangleq \deg \Lambda_v(x) + k,$$

and where $\hat{E}(x) \triangleq \Lambda_v(x)E(x) \mod m(x)$. Clearly,

$$\Lambda_u(x)\hat{E}(x) \mod m(x) = 0.$$
Theorem 2.7  If $\deg \Lambda_u(x) \leq \frac{n-k}{2}$, then
$$\deg(\hat{Y}(x)\Lambda_u(x) \mod m(x)) < (n + \hat{k})/2$$  \hspace{1cm} (2.137)

Conversely, for $t \in \mathbb{R}$ with
$$\deg \Lambda_u(x) \leq t \leq (n - \hat{k})/2,$$  \hspace{1cm} (2.138)
if some nonzero $\Lambda(x) \in F[x]$ with $\deg \Lambda(x) \leq t$ satisfies
$$\deg\left(\hat{Y}(x)\Lambda(x) \mod m(x)\right) < n - t,$$  \hspace{1cm} (2.139)
then $\Lambda(x)$ is a multiple of $\Lambda_u(x)$.

The theorem is an exact analog of Theorem 2.3; its proof is omitted here since the same argument as in Section 2.5 applies. We thus have the decoding procedure as in Section 2.5: compute $\hat{Y}(x)$, and run the PIA of Section 2.3 with $b(x) = \hat{Y}(x)$, and $d = \lceil \frac{n+k}{2} \rceil$. If
$$\deg \Lambda_u(x) \leq (n - \hat{k})/2,$$  \hspace{1cm} (2.140)
which coincides with (2.132), then the polynomial returned by PIA will equal $\gamma\Lambda_u(x)$ for some nonzero $\gamma \in F$. We then complete decoding by (2.29) with $\Lambda(x) = \gamma\Lambda_u(x)\Lambda_v(x)$.

Approach-II

This approach will not require the pre-computation of (2.133). Instead, we will use the IC-PIA of Section 2.11 with $b(x) = Y(x)$, $\Lambda_c(x) = \Lambda_u(x)$ and $d = \lceil \frac{n+k}{2} \rceil$ to obtain $\Lambda(x) = \gamma\Lambda_u(x)\Lambda_v(x)$ directly.

The idea is to translate the problem of solving (2.139) into the CPI problem of Section 2.11. Indeed, $\hat{Y}(x)\Lambda(x) \mod m(x)$ can be written
$$\hat{Y}(x)\Lambda(x) \mod m(x) = Y(x)(\Lambda(x)\Lambda_v(x)) \mod m(x).$$

Note also that $\gcd(\Lambda_u(x), \Lambda_v(x)) = 1$. The following theorem then follows from Theorem 2.7 by setting $t = (n - \hat{k})/2$:

Theorem 2.8  For some nonzero $\Lambda(x) \in F[x]$ that is a multiple of $\Lambda_v(x)$ and has degree $\deg \Lambda(x) \leq \deg \Lambda_v(x) + (n - \hat{k})/2$, if $\Lambda_u(x) \leq (n - \hat{k})/2$ and if
$$\deg\left(\hat{Y}(x)\Lambda(x) \mod m(x)\right) < (n + \hat{k})/2,$$  \hspace{1cm} (2.141)
then $\Lambda(x)$ is also a multiple of $\Lambda_u(x)$. 

We can then get $\Lambda(x) = \gamma \Lambda_u(x) \Lambda_v(x)$ directly by running the IC-PIA of Section 2.11 with $b(x) = Y(x)$, $\Lambda_c(x) = \Lambda_v(x)$ and $d = \lceil \frac{n+k}{2} \rceil$. If (2.132) is satisfied, then the $\Lambda(x)$ returned by IC-PIA will equal to $\Lambda_u(x) \Lambda_v(x)$ up to a scale factor.

**Approach-III**

This approach is essentially a complement to Approach-I. Let

$$\tilde{m}(x) \triangleq m(x)/\Lambda_v(x),$$

which has degree

$$\deg \tilde{m}(x) \triangleq \tilde{n} = n - \deg \Lambda_v(x),$$

and let

$$\tilde{Y}(x) \triangleq Y(x) \mod \tilde{m}(x).$$

We can then run the PIA of Section 2.3 with the corresponding $\tilde{Y}(x)$, $\tilde{m}(x)$ and $d = \lceil \frac{\tilde{n}+k}{2} \rceil$ as inputs. If $\deg \Lambda_u(x) \leq (\tilde{n} - k)/2$, which coincides with (2.132), the PIA will return $\Lambda(x) = \gamma \Lambda_u(x)$ for some nonzero $\gamma \in F$; we then complete decoding (as in Proposition 2.8) by

$$C(x) = \frac{Y(x) \Lambda(x) \mod \tilde{m}(x)}{\Lambda(x)}$$

Note that the PIA of Approach-III has inputs of lower degrees than the PIA employed by Approach-I, i.e., $\deg \tilde{m}(x) < \deg m(x)$ and $\deg \tilde{Y}(x) < \deg \tilde{Y}(x)$, provided that $\deg \Lambda_v(x) \geq 1$.

**Remark**

The ideas of Approaches-I and II are similar to those in [10, 22, 23] that use either the Berlekamp-Massey algorithm or the Euclidean algorithm for joint errors-and-erasures decoding of Reed-Solomon codes. However, the idea of Approach-III (see also [8]) appears to be new.

### 2.13 Conclusion

We have proposed a new algorithm (PIA) together with an extended algorithm (QSA) to find a nonzero polynomial $\Lambda(x)$ of the smallest degree that satisfies

$$\deg \left( b(x) \Lambda(x) \mod m(x) \right) < d$$

(2.146)
for any given \( b(x) \) and \( m(x) \neq 0 \) with \( \deg b(x) \leq \deg m(x) \), and for any given \( d \) with \( 1 \leq d \leq \deg m(x) \). We showed that computing inverses in \( F[x]/m(x) \), computing Padé approximants, and decoding (generalized) Reed-Solomon codes (either by solving the standard or by the alternative key equation) can all be reduced to this problem.

The proposed algorithm **PIA** (and/or **QSA**) can be carried out in \( O(d_m^2) \) operations, where \( d_m \triangleq \deg m(x) \). In the special case where \( m(x) = x^n \) or \( m(x) = x^n - 1 \), the algorithm **PIA** almost coincides with the Berlekamp-Massey algorithm, except that it processes \( b(x) \) in reverse order. However, the proposed **PIA** (and/or **QSA**) works naturally for general \( m(x) \); indeed, as we have shown, **QSA** works for general \( m(x) \) in explicitly the same fashion (with the same style of computation) as the Berlekamp-Massey algorithm.

We also made an easy translation that turns **PIA** into the **RSA**, a version (without scalar inversion) of the Euclidean algorithm. Together with the introduced notion of minimal partial-inverse, such a translation also contributes a different insight into the Euclidean algorithm. Another consequence is that the proposed **PIA** and **QSA** possess the computational advantage as the Berlekamp-Massey algorithm, while keeping also the functional advantage of the Euclidean algorithm.

Finally, we conclude that the formulation of the partial-inverse problem provides a simple and unified view to the many applications.
Chapter 3

Chinese Remainder and Polynomial Remainder Codes

The partial-inverse approach of Chapter 2 is generalized to decoding polynomial remainder codes, a large class of codes that include Reed-Solomon codes as a special case.

The chapter begins with the Chinese remainder theorem, which leads to a natural definition of the Chinese remainder codes that include polynomial remainder codes. Of particular interest are the two interpolation which work for the Chinese remainder codes. Moreover, the theory of polynomial remainder codes are carefully constructed as in prior work. In particular, code symbols of such codes are explicitly allowed to be polynomials of different degrees; moreover, if the moduli are not irreducible, the notion of an error locator polynomial is replaced by the notion of an error factor polynomial.

3.1 Introduction

Polynomial remainder codes are a large class of codes derived from the Chinese remainder theorem. Such codes were proposed by Stone [24], who also pointed out that these codes include Reed-Solomon codes [1] as a special case. Variations of Stone’s codes were studied in [25, 27]. In [24] and [25], the focus is on codes with a fixed symbol size, i.e., the moduli
are relatively prime polynomials of the same degree. A generalization of such codes was proposed by Mandelbaum, who also pointed out that using moduli of different degrees can be advantageous for burst error correction.

Although the codes in [24–27] can, in principle, correct many random errors, no efficient decoding algorithm for random errors was proposed in these papers. In 1988, Shiozaki proposed an efficient decoding algorithm for Stone’s codes using Euclid’s algorithm, and he also adapted this algorithm to decode Reed-Solomon codes. However, the algorithm of [19] is restricted to codes with a fixed symbol size, i.e., fixed-degree moduli. Moreover, the argument given in [19] seems to assume that all the moduli are irreducible although this assumption is not stated explicitly.

In [28], Mandelbaum made the interesting observation that polynomial remainder codes (generalized as in [26]) contain Goppa codes as a special case. By means of this observation, generalized versions of Goppa codes such as in [30] may also be viewed as polynomial remainder codes. In subsequent work [31,32], Mandelbaum actually used the term “generalized Goppa codes” for (generalized) polynomial remainder codes. He also proposed a decoding algorithm for such codes using a continued-fractions approach. However, this connection between (generalized) polynomial remainder codes and Goppa codes will not be further pursued in this thesis.

There is also a body of work on Chinese remainder codes over integers, cf. [33,34]. However, the results of the present chapter are not directly related to that work.

In this chapter, we revisit polynomial remainder codes as in [24]. We explicitly allow moduli of different degrees (i.e., variable symbol sizes) within a codeword. In this way, we can, e.g., lengthen a Reed-Solomon code by adding some higher-degree symbols without increasing the size of the underlying field. In consequence, we obtain two different notions of distance—Hamming distance and degree-weighted distance—and the corresponding minimum-distance decoding rules. Recall from Section 2.5 that decoding Reed-Solomon codes can be reduced rather directly to the Partial-Inverse Problem of Section 2.1. We are going to propose such an approach for polynomial remainder codes. As we will see later, if the moduli are not irreducible, the notion of an error locator polynomial is replaced by an error factor polynomial.

This chapter is organized as follows. In Section 3.2 we recall the Chinese remainder theorem and the definition of Chinese remainder codes.
3.2 Chinese Remainder Codes

3.2.1 Chinese Remainder Theorem and Codes

Let $R = \mathbb{Z}$ or $R = \mathbb{F}[x]$ for some field $\mathbb{F}$. (Later on, we will focus on $R = \mathbb{F}[x]$.) For $R = \mathbb{Z}$, for any positive $m \in \mathbb{Z}$, let $\mathbb{Z}_m$ denote the ring $\{0, 1, 2, \ldots, m - 1\}$ with addition and multiplication modulo $m$; for $R = \mathbb{F}[x]$, for any monic polynomial $m(x) \in \mathbb{F}[x]$, let $\mathbb{F}[x]_m$ denote the ring of polynomials over $\mathbb{F}$ of degree less than $\deg m(x)$ with addition and multiplication modulo $m(x)$. For $R = \mathbb{Z}$, $\gcd(a, b)$ denotes the greatest common divisor of $a, b \in \mathbb{Z}$, not both zero; for $R = \mathbb{F}[x]$, $\gcd(a, b)$ denotes
the monic polynomial of largest degree that divides both \( a, b \in F[x] \), not both zero.

We will need the Chinese remainder theorem \(^\square\) in the following form.

**Theorem 3.1 (Chinese Remainder Theorem)** For some integer \( n > 1 \), let \( m_0, m_1, \ldots, m_{n-1} \in R \) be relatively prime (i.e., \( \gcd(m_i, m_j) = 1 \) for \( i \neq j \)) and let \( M_n \triangleq \prod_{i=0}^{n-1} m_i \). Then the mapping

\[
\psi : R_{M_n} \to R_{m_0} \times \cdots \times R_{m_{n-1}} : a \mapsto (\psi_0(a), \ldots, \psi_{n-1}(a)) \tag{3.1}
\]

with \( \psi_i(a) \triangleq a \mod m_i \) is a ring isomorphism.

The inverse of the mapping \((3.1)\) is

\[
\psi^{-1} : R_{m_0} \times \cdots \times R_{m_{n-1}} \to R_{M_n} : (c_0, \ldots, c_{n-1}) \mapsto \sum_{i=0}^{n-1} c_i \beta_i \mod M_n \tag{3.2}
\]

with coefficients

\[
\beta_i = \frac{M_n}{m_i} \cdot \left( \frac{M_n}{m_i} \right)^{-1} \mod m_i \tag{3.3}
\]

where \((b)^{-1} \mod m_i\) denotes the inverse of \( b \) in \( R_{m_i} \).

**Definition 3.1** A Chinese remainder code (CRT Code) over \( R \) is a set of the form

\[
C \triangleq \{(c_0, \ldots, c_{n-1}) : c_i = a \mod m_i \text{ for some } a \in R_{M_k}\} \tag{3.4}
\]

where \( n \) and \( k \) are integers satisfying \( 1 \leq k \leq n \), where \( m_0, \ldots, m_{n-1} \in R \) are relatively prime, and where \( M_k \triangleq \prod_{i=0}^{k-1} m_i \). \( \square \)

In other words, a CRT code consists of the images \( \psi(a) \), with \( \psi \) as in \((3.1)\), of all \( a \in R_{M_k} \). For \( R = F[x] \), CRT codes are linear (i.e., vector spaces) over \( F \); for \( R = \mathbb{Z} \), however, CRT codes are not linear since the pre-image of the sum of two codewords may exceed the range of \( M_k \).

The components \( c_i = \psi_i(a) \) in \((3.1)\) and \((3.4)\) will be called symbols. Note that each symbol is from a different ring \( R_{m_i} \); these rings need not have the same number of elements. We will often (but not always) assume that the moduli \( m_i \) in Definition \((3.1)\) satisfy the condition

\[
|R_{m_0}| \leq |R_{m_1}| \leq \cdots \leq |R_{m_{n-1}}| \tag{3.5}
\]

We will refer to \((3.5)\) as the Ordered-Symbol-Size Condition.
3.2 Chinese Remainder Codes

3.2.2 Interpolation

Consider the problem of reconstructing a codeword \( c = (c_0, \ldots, c_{n-1}) \) from a subset of its symbols. Specifically, let \( C \) be a CRT code as in Definition 3.1 and let \( S \) be a subset of \( \{0, 1, 2, \ldots, n-1\} \) with cardinality \( |S| > 0 \). Let \( c = (c_0, \ldots, c_{n-1}) = \psi(a) \in C \) be the codeword corresponding to some \( a \in R_{M_k} \) by (3.4). Suppose we are given \( \tilde{c} = (\tilde{c}_0, \ldots, \tilde{c}_{n-1}) \) with

\[
\tilde{c}_i = c_i \text{ for } i \in S
\]

(and with arbitrary \( \tilde{c}_i \in R_{m_i} \) for \( i \notin S \)) and we wish to reconstruct \( a = \psi^{-1}(c) \) from \( \tilde{c} \). This problem arises, for example, when the channel erases some symbols (and lets the receiver know the erased positions) but delivers the other symbols unchanged. However, this problem also arises as the last step in the decoding procedures that will be discussed later in the chapter.

This interpolation problem can certainly be solved if \( S \) is sufficiently large. A first solution follows immediately from the CRT (Theorem 3.1). Specifically, with \( M_S \triangleq \prod_{i \in S} m_i \), Theorem 3.1 can be applied as follows: if

\[
|M_S| \geq |M_k|
\]

then

\[
a = \sum_{i=0}^{n-1} \tilde{c}_i \tilde{\beta}_i \mod M_S
\]

with

\[
\tilde{\beta}_i = \begin{cases} 
\frac{M_S}{m_i} \cdot \left( \frac{M_S}{m_i} \right)^{-1} \mod m_i, & i \in S \\
0, & i \notin S.
\end{cases}
\]

Obviously, the coefficients \( \tilde{\beta}_i \) in (3.9) depend on the support set \( S \). Interestingly, there are two solutions to the interpolation problem, both of which avoid the computation of these coefficients: the two theorems below show how \( a = \psi^{-1}(c) \) can be computed from \( \psi^{-1}(\tilde{c}) \), which in turn may be computed using the fixed coefficients (3.3).

**Theorem 3.2 (Fixed-Transform Interpolation I)** If

\[
|M_S| \geq |M_k|
\]

then

\[
\psi^{-1}(c) = Z/M_S
\]
where \( M_\mathcal{S} \triangleq M_n / M_S \) and where

\[
Z \triangleq (M_\mathcal{S} \cdot \psi^{-1}(\tilde{c})) \mod M_n
\]  

(3.12)
is a multiple of \( M_\mathcal{S} \).

**Proof:** Let \( \tilde{c} \triangleq c - \tilde{c} \), let \( \tilde{a} \triangleq \psi^{-1}(\tilde{c}) \), and note that \( \psi^{-1}(\tilde{c}) = (a - \tilde{a}) \mod M_n \). Note also that \( |M_\mathcal{S} \cdot a| < |M_n| \) because of (3.10). Then

\[
Z = (M_\mathcal{S} \cdot (a - \tilde{a})) \mod M_n
\]  

(3.13)

\[
= M_\mathcal{S} \cdot a - (M_\mathcal{S} \cdot \tilde{a}) \mod M_n
\]  

(3.14)

\[
= M_\mathcal{S} \cdot a
\]  

(3.15)

where the last step follows from

\[
\psi(M_\mathcal{S} \cdot \tilde{a}) = \psi(M_\mathcal{S}) \psi(\tilde{a})
\]  

(3.16)

\[
= 0.
\]  

(3.17)

\[ \square \]

**Theorem 3.3 (Fixed-Transform Interpolation II)** If

\[
|M_S| \geq |M_k|
\]  

(3.18)
then

\[
\psi^{-1}(c) = \psi^{-1}(\tilde{c}) \mod M_S
\]  

(3.19)

**Proof:** Let \( \tilde{c} \triangleq c - \tilde{c} \), and let \( \tilde{a} \triangleq \psi^{-1}(\tilde{c}) \). Note from (3.6) that \( \tilde{a} \mod m_\mathcal{S} = 0 \) for every \( i \in S \), and thus \( \tilde{a} \mod M_S = 0 \). Note also that \( \psi^{-1}(\tilde{c}) = (a - \tilde{a}) \mod M_n \) where \( a = \psi^{-1}(c) \) with \( |a| < |M_k| \). Then

\[
\psi^{-1}(\tilde{c}) \mod M_S = ((a - \tilde{a}) \mod M_n) \mod M_S
\]  

(3.20)

\[
= (a - \tilde{a}) \mod M_S
\]  

(3.21)

\[
= a
\]  

(3.22)

where the last step follows from \( \tilde{a} \mod M_S = 0 \) and (3.18). \[ \square \]

Both Theorems 3.2 and 3.3 do not appear in standard expositions of the CRT; perhaps they are new. Their applications to coding, even to Reed-Solomon codes (cf. Propositions 2.8 and 2.9 of Section 2.5 or cf. Section 3.3.3), also appear to be new.
3.2.3 Hamming Distance and Singleton Bound

For any \( a \in \mathbb{R}_{M_n} \), the Hamming weight of \( \psi(a) \) (i.e., the number of nonzero symbols \( \psi_i(a) \), \( 0 \leq i \leq n-1 \)) will be denoted by \( w_H(\psi(a)) \). For any \( a, b \in \mathbb{R}_{M_n} \), the Hamming distance between \( \psi(a) \) and \( \psi(b) \) will be denoted by \( d_H(\psi(a), \psi(b)) \triangleq w_H(\psi(a) - \psi(b)) \). The minimum Hamming distance of a CRT code \( C \) will be denoted by \( d_{\text{min}}(C) \).

**Theorem 3.4** Let \( C \) be a CRT code as in Definition 3.1 satisfying the Ordered-Symbol-Size Condition (3.5). Then the Hamming weight of any nonzero codeword \( \psi(a) \) (\( a \in \mathbb{R}_{M_k}, a \neq 0 \)) satisfies

\[
w_H(\psi(a)) \geq n - k + 1 \quad (3.23)
\]

and

\[
d_{\text{min}}(C) = n - k + 1. \quad (3.24)
\]

**Proof:** For any nonzero \( a \in \mathbb{R}_{M_n} \), assume that the image \( \psi(a) \) has Hamming weight \( w_H(\psi(a)) \leq n - k \), i.e., the number of zero symbols of \( \psi(a) \) is at least \( k \). For \( R = \mathbb{Z} \), this implies \( a \geq M_k \); for \( R = \mathbb{F}[x] \), this implies \( \deg a \geq \deg M_k \). In both cases, \( a \not\in \mathbb{R}_{M_k} \), which proves (3.23).

As for (3.24), consider \( d_H(\psi(a), \psi(b)) \) for any \( a, b \in \mathbb{R}_{M_k}, a \neq b \). For \( R = \mathbb{F}[x] \), \( a - b \in \mathbb{R}_{M_k} \) and thus

\[
d_H(\psi(a), \psi(b)) = w_H(\psi(a) - \psi(b)) \quad (3.25)
\]

\[
= w_H(\psi(a - b)) \quad (3.26)
\]

\[
\geq n - k + 1 \quad (3.27)
\]

by (3.23). For \( R = \mathbb{Z} \), either \( a - b \in \mathbb{R}_{M_k} \) or \( b - a \in \mathbb{R}_{M_k} \) and the same argument applies. It follows that \( d_{\text{min}}(C) \geq n - k + 1 \). Finally, the equality in (3.24) follows from the Singleton bound below. \( \square \)

In the following theorem, we will use the following notation. For any subset \( S \subset \{0, 1, \ldots, n-1\} \), let \( \mathcal{S} \triangleq \{0, 1, \ldots, n-1\} \setminus S \) and let

\[
R_S \triangleq \bigotimes_{i \in S} \mathbb{R}_{m_i}, \quad (3.28)
\]

the direct product of all rings \( \mathbb{R}_{m_i} \) with \( i \in S \).
Theorem 3.5 (Singleton Bound for Hamming Distance) Let $C$ be a code in $R_{\{0, \ldots, n-1\}}$ (i.e., a nonempty subset of $R_{m_0} \times \cdots \times R_{m_{n-1}}$) with minimum Hamming distance $d_{\text{minH}}$. Then

$$|C| \leq \min_{S \subseteq \{0, 1, \ldots, n-1\}} \{|R_S| : |S| > n - d_{\text{minH}}\}.$$ (3.29)

**Proof:** Let $S$ be a subset of $\{0, 1, \ldots, n-1\}$ with $|S| < d_{\text{minH}}$. For every word $c \in C$, erase its components in $S$. The resulting set of shortened words, which are elements of $R_S$, has still $|C|$ elements. 

Note that this theorem does not require the Ordered-Symbol-Size Condition (3.5).

For CRT codes satisfying the Ordered-Symbol-Size Condition (3.5), we have $|C| = |R_{M_k}|$; on the other hand, the right-hand side of (3.29) becomes

$$|R_{\{0, \ldots, n-d_{\text{minH}}\}}| = |R_{M_{n-d_{\text{minH}}+1}}|$$ (3.30)

where $M_{n-d_{\text{minH}}+1} \triangleq \prod_{i=0}^{n-d_{\text{minH}}} m_i$. It then follows from (3.29) that $|R_{M_k}| \leq |R_{M_{n-d_{\text{minH}}+1}}|$ and thus

$$k \leq n - d_{\text{minH}} + 1.$$ (3.31)

### 3.3 Polynomial Remainder Codes (PRC)

From now on, we will focus on the case $R = F[x]$ for some finite field $F$.

#### 3.3.1 Definition and Some Examples

**Definition 3.2** A polynomial remainder code is a CRT code over $R = F[x]$ with monic moduli $m_i(x)$, i.e., a set of the form

$$C = \{(c_0, \ldots, c_{n-1}) : c_i = a(x) \mod m_i(x) \text{ for some } a(x) \in R_{M_k}\}.$$ (3.32)

A polynomial remainder code is irreducible if the polynomials $m_0(x)$, $m_1(x)$, $\ldots$, $m_{n-1}(x)$ are all irreducible. 

For such codes, the Ordered-Symbol-Size Condition (3.5) may be written as

$$\deg m_0(x) \leq \deg m_1(x) \leq \ldots \leq \deg m_{n-1}(x),$$ (3.33)

which we will call the Ordered-Degree Condition.
Example 3.1: Binary Irreducible Polynomial Remainder Codes
Let \( F = \text{GF}(2) \) be the finite field with two elements and let \( m_0(x), m_1(x), \ldots, m_{n-1}(x) \) be different irreducible binary polynomials.

The number of irreducible binary polynomials of degree up to 16 is given in Appendix A. For example, by using only irreducible moduli of degree 16, we can obtain a code with \( \deg M_n(x) = 4080 \); by using irreducible moduli of degree up to 16, we can achieve \( \deg M_n(x) = 130'486 \).

Example 3.2: Polynomial Evaluation Codes and Reed-Solomon Codes
Let \( \beta_0, \beta_1, \ldots, \beta_{n-1} \) be distinct elements of some finite field \( F \) (which implies \( n \leq |F| \)). A polynomial evaluation code over \( F \) is a code of the form
\[
C \triangleq \{(c_0, \ldots, c_{n-1}) : c_i = a(\beta_i) \text{ for some } a(x) \in F[x] \text{ of } \deg a(x) < k\}.
\]
(3.34)
A Reed-Solomon code is a polynomial evaluation code with \( \beta_i = \alpha^i \), where \( \alpha \) is a primitive \( n \)-th root of unity in \( F \). With
\[
m_i(x) \triangleq x - \beta_i,
\]
(3.35)
a polynomial evaluation code may be viewed as a polynomial remainder code since
\[
c_i = a(\beta_i) = a(x) \mod m_i(x).
\]
(3.36)
For Reed-Solomon codes (as defined above), we then have
\[
M_n(x) = x^n - 1.
\]
(3.37)

Example 3.3: Polynomial Extensions of Reed-Solomon Codes
When Reed-Solomon codes are viewed as polynomial remainder codes as in Example 3.2, the code symbols are constants, i.e., polynomials of degree at most zero. Reed-Solomon codes can be extended with additional symbols in \( F[x] \) by adding some moduli \( m_i(x) \) of degree two (or higher).

3.3.2 Degree-weighted Distance
Let
\[
N \triangleq \deg M_n(x) = \sum_{i=0}^{n-1} \deg m_i(x)
\]
(3.38)
and
\[ K \triangleq \deg M_k(x) = \sum_{i=0}^{k-1} \deg m_i(x). \] (3.39)

Note that \( K \) is the dimension of the code as a subspace of \( F^N \).

**Definition 3.3** The *degree weight* of a set \( S \subset \{0, 1, \ldots, n-1\} \) is
\[ w_D(S) \triangleq \sum_{i \in S} \deg m_i(x). \] (3.40)

For any \( a(x) \in R_{M_n} \), the degree weight of \( \psi(a) = (\psi_0(a), \ldots, \psi_{n-1}(a)) \) is
\[ w_D(\psi(a)) \triangleq \sum_{i : \psi_i(a) \neq 0} \deg m_i, \] (3.41)
and for any \( a(x), b(x) \in R_{M_n} \), the *degree-weighted distance* between \( \psi(a) \) and \( \psi(b) \) is
\[ d_D(\psi(a), \psi(b)) \triangleq w_D(\psi(a) - \psi(b)). \] (3.42)

Note that the degree-weighted distance satisfies the triangle inequality:
\[ d_D(\psi(a), \psi(b)) \leq d_D(\psi(a), \psi(c)) + d_D(\psi(b), \psi(c)) \] (3.43)
for all \( a(x), b(x), c(x) \in R_{M_n} \).

Let \( d_{\min D}(C) \) denote the *minimum degree-weighted distance* of a polynomial remainder code \( C \), i.e.,
\[ d_{\min D}(C) \triangleq \min_{c, c' \in C : c \neq c'} d_D(c, c'), \] (3.44)
and let
\[ w_{\min D}(C) \triangleq \min_{c \in C : c \neq 0} w_D(c) \] (3.45)
be the minimum degree weight of any nonzero codeword. We then have the following analog of Theorem 3.4:

**Theorem 3.6 (Minimum Degree-Weighted Distance)** Let \( C \) be a code as in Definition 3.2. Then
\[ d_{\min D}(C) = w_{\min D}(C) \] (3.46)
\[ = \min_{S \subset \{0, \ldots, n-1\}} \left\{ w_D(S) : w_D(S) > N - K \right\} \] (3.47)
\[ > N - K. \] (3.48)
If all moduli \( m_i(x) \) have degree one, then the right-hand side of (3.47) equals \( N - K + 1 \). Note also that unlike Theorem 3.4, Theorem 3.6 does not require the Ordered-Degree Condition (3.33).

**Proof of Theorem 3.6:** Equation (3.46) is obvious from the linearity of the code over \( F \), and (3.48) is obvious as well. It remains to prove (3.47).

Let \( d \) be the right-hand side of (3.47). For any nonzero \( a(x) \in R_{M_k} \), assume that the image \( \psi(a) \) has degree weight \( w_D(\psi(a)) \leq N - K \), i.e., the sum of \( \deg m_i(x) \) over the zero symbols of \( \psi(a) \) is at least \( K \). Then \( \deg a(x) \geq K = \deg M_k(x) \), which is impossible since \( a(x) \in R_{M_k} \). We thus have \( w_D(\psi(a)) > N - K \). It then follows from Definition 3.3 that \( w_D(\psi(a)) \geq d \) and thus \( w_{\min D}(C) \geq d \).

Conversely, let \( S \) be a subset of \( \{0, 1, \ldots, n-1\} \) such that \( w_D(S) = d \). Then there exists some nonzero \( a(x) \in R_{M_k} \) such that \( \psi_i(a) \neq 0 \) for each \( i \in S \) but \( \psi_j(a) = 0 \) for each \( j \in \{0, 1, \ldots, n-1\} \setminus S \). Thus \( w_D(\psi(a)) = w_D(S) = d \), which implies \( w_{\min D}(C) \leq d \). □

**Theorem 3.7 (Singleton Bound for Degree-weighted Distance)**

Let \( C \) be a nonempty subset of \( R_{m_0} \times \cdots \times R_{m_{n-1}} \) with minimum degree-weighted distance \( d_{\min D} \) and with \( N \) as in (3.38). Then

\[
\log_F |C| \leq \min_{S \subseteq \{0, \ldots, n-1\}} \{ w_D(S) : w_D(S) > N - d_{\min D} \}. \tag{3.49}
\]

**Proof:** Recall the notation \( \overline{S} \) and \( R_S \) as in (3.28). Let \( \overline{S} \) be a subset of \( \{0, 1, \ldots, n-1\} \) with \( w_D(\overline{S}) < d_{\min D} \). For every word \( c \in C \), erase its components in \( \overline{S} \). The resulting set of shortened words, which are elements of \( R_S \), has still \( |C| \) elements. Thus \( |C| \leq |R_S| = |F|^{w_D(\overline{S})} \), and (3.49) follows. □

For polynomial remainder codes, we have \( \log_F |C| = K \) and (3.49) holds with equality. To see this, we first write (3.49) as

\[
K \leq \min_{S \subseteq \{0, \ldots, n-1\}} \{ w_D(S) : w_D(S) > N - d_{\min D} \}. \tag{3.50}
\]

On the other hand, for \( S = \{0, \ldots, k-1\} \), we have \( w_D(S) = K \), and using (3.48), we obtain

\[
\min_{S \subseteq \{0, \ldots, n-1\}} \{ w_D(S) : w_D(S) > N - d_{\min D} \} \leq K. \tag{3.51}
\]

We thus have equality in (3.50) and (3.51), and therefore also in (3.49).

In the special case where all the moduli \( m_0(x), \ldots, m_{n-1}(x) \) have the same degree, the two Singleton bounds (3.49) and (3.29) are equivalent.
3.3.3 Interpolation and Erasures Decoding

We now return to the subject of Section 3.2.2 and specialize it to polynomial remainder codes. Let $C$ be a code as in Definition 3.2. Let $c = (c_0, \ldots, c_{n-1}) = \psi(a(x)) \in C$ be the codeword corresponding to some polynomial $a(x) \in \mathbb{R}_{M_k}$. Let $S$ be a set of positions $i \in \{0, \ldots, n-1\}$ where $c_i$ is known. Let $\tilde{c} = (\tilde{c}_0, \ldots, \tilde{c}_{n-1})$ satisfy $\tilde{c}_i = c_i$ for $i \in S$ with arbitrary $\tilde{c}_i \in \mathbb{R}_{m_i}$ for $i \notin S$. Suppose we wish to reconstruct $a(x)$ from $\tilde{c}$ and $S$.

Let $\overline{S} \triangleq \{0, \ldots, n-1\} \setminus S$ be the indices of the unknown components of $c$ and let $M_{\overline{S}}(x) = \prod_{i \in \overline{S}} m_i(x)$ as in Section 3.2.2. Recall that $w_D(\overline{S})$ denotes the degree weight of the unknown (erased) components of $c$. Then Theorem 3.2 can be restated as follows:

**Theorem 3.8 (Fixed-Transform Interpolation I for PRC)** If

$$w_D(\overline{S}) \leq N - K,$$

then

$$a(x) = Z(x)/M_{\overline{S}}(x)$$

with

$$Z(x) \triangleq M_{\overline{S}}(x) \psi^{-1}(\tilde{c}) \mod M_n(x).$$

The equivalence of (3.52) and (3.10) follows from noting that the left-hand side of (3.10) is $|M_{\overline{S}}| = N - w_D(\overline{S})$ and the right-hand side of (3.10) is $|M_k| = K$.

Since $\overline{S}$ contains the support set of $\tilde{c} - c$, the polynomial $M_{\overline{S}}(x)$ is a multiple of an error locator polynomial (as will be defined in Section 3.4).

Theorem 3.8 appears to be new even when specialized to Reed-Solomon codes (as in Example 3.2), where $M_n(x) = x^n - 1$ and the modulo operation in (3.54) is computationally trivial.

Finally, we note that in this respect, Theorem 3.3 reads as follows:

**Theorem 3.9 (Fixed-Transform Interpolation II for PRC)** If

$$w_D(\overline{S}) \leq N - K,$$

then

$$a(x) = \psi^{-1}(\tilde{c}) \mod M_S(x)$$

where $M_S(x) = \prod_{i \in S} m_i(x)$. 
3.3.4 Minimum-Distance Decoding

Let $C$ be a code as in Definition 3.2. The receiver sees $y = c + e$, where $c \in C$ is the transmitted codeword and $e$ is an error pattern. A minimum Hamming distance decoder is a decoder that produces

$$\hat{c} = \arg\min_{c \in C} d_H(c, y).$$

(3.57)

A minimum degree-weighted distance decoder is a decoder that produces

$$\hat{c} = \arg\min_{c \in C} d_D(c, y).$$

(3.58)

In general, the decoding rules (3.57) and (3.58) produce different estimates $\hat{c}$ as will be illustrated by the examples below.

Theorem 3.10 (Basic Error Correction Bounds) If $d_H(c, y) < d_{\min H}(C)/2$, then (3.57) produces $\hat{c} = c$. If $d_D(c, y) < d_{\min D}(C)/2$, then the rule (3.58) produces $\hat{c} = c$.

Proof: The proof follows the standard pattern; we prove only the second part. Assume $\hat{c} \neq c$, which implies $d_D(\hat{c}, y) \leq d_D(c, y)$. Using the triangle inequality (3.43), we obtain $d_{\min D}(C) \leq d_D(\hat{c}, c) \leq d_D(\hat{c}, y) + d_D(c, y) \leq 2d_D(c, y)$. □

The second part of Theorem 3.10 can also be formulated as follows: if

$$w_D(e) \leq t_D \triangleq \left\lfloor \frac{N - K}{2} \right\rfloor,$$

(3.59)

then the rule (3.58) produces $\hat{c} = c$. If the Ordered-Degree Condition (3.33) is satisfied, then the first part of Theorem 3.10 implies the following: if

$$w_H(e) \leq t_H \triangleq \left\lfloor \frac{n - k}{2} \right\rfloor,$$

(3.60)

then the rule (3.57) produces $\hat{c} = c$.

Depending on the degrees $\deg m_i(x)$, it is possible that the condition $w_H(e) \leq t_H$ implies $w_D(e) \leq t_D$ (see Example 3.5 below). In general, however, none of the two decoding rules (3.57) and (3.58) is uniformly stronger than the other.
Example 3.4
Let \( k = 3 \) and \( n = 5 \), and let \( \deg m_i(x) = i \) for \( i = 1, 2, \ldots, 5 \). We then have \( t_H = 1 \), \( K = 6 \), \( N = 15 \), and \( t_D = 4 \). Consider the following two decoders: Decoder A corrects all errors with \( w_H(e) \leq t_H \) and Decoder B corrects all errors with \( w_D(e) \leq t_D \). We then observe:

- Decoder A corrects all single symbol errors in any position.
- Decoder B corrects all single symbol errors in the first 4 symbols (but not in position 5), and it corrects two symbol errors in positions 1 and 2, or in positions 1 and 3.

Example 3.5
Let \( k = 3 \) and \( n = 5 \), and let \( \deg m_1(x) = \deg m_2(x) = \deg m_3(x) = 1 \) and \( \deg m_4(x) = \deg m_5(x) = 2 \). We then have \( t_H = 1 \), \( K = 3 \), \( N = 7 \), and \( t_D = 2 \). Considering the same decoders as in Example 3.4, we observe:

- Decoder A corrects all single symbol errors in any position.
- Decoder B also corrects all single symbol errors, and in addition, it corrects any two symbol errors in the first 3 symbols.

3.3.5 Summary of Code Parameters

Let us summarize the key parameters of a polynomial remainder code \( C \) both in terms of Hamming distance and in terms of degree-weighted distance. For the latter, the code parameters are \((N, K, d_{\text{minD}})\) with \( N \), \( K \), and \( d_{\text{minD}} \) defined as in (3.38), (3.39), (3.44) and with \( d_{\text{minD}} \) as in (3.47). By the rate of the code, we mean the quantity

\[
\frac{1}{N} \log_2 |F| |C| = \frac{K}{N} \quad (3.61)
\]

where \( F \) is the underlying field.

With respect to Hamming distance, we have the code parameters \((n, k, d_{\text{minH}})\) and the symbol rate \( k/n \). If the code \( C \) satisfies the Ordered-Degree Condition (3.33), we have \( d_{\text{minH}} = n - k + 1 \).

In the special case where all the moduli \( m_0(x), \ldots, m_{n-1}(x) \) have the same degree, the two triples \((N, K, d_{\text{minD}})\) and \((n, k, d_{\text{minH}})\) are equal up to a scale factor and the rate (3.61) equals the symbol rate \( k/n \).
Decoding polynomial remainder codes can be reduced to solving a key equation as in Section 2.5. But for such codes, we will need a notion of error factor polynomial.

Let $C$ be a polynomial remainder code of the form (3.32). For the received $y = c + e$, where $c = (c_0, \ldots, c_{n-1}) \in C$ is a transmitted codeword, and where $e = (e_0, \ldots, e_{n-1})$ is an error pattern, let $Y(x) = a(x) + E(x)$ denote the pre-image $\psi^{-1}(y)$ of $y$ with $\psi^{-1}$ as in (3.2), where $a(x) = \psi^{-1}(c)$ is the transmitted-message polynomial, and where $E(x)$ denotes the pre-image $\psi^{-1}(e)$ of the error $e$.

**Definition 3.4** An error factor polynomial is a nonzero polynomial $\Lambda(x) \in F[x]$ such that

$$\Lambda(x)E(x) \mod M_n(x) = 0. \tag{3.62}$$

Clearly, the polynomial

$$\Lambda_f(x) \triangleq \frac{M_n(x)}{\gcd (E(x), M_n(x))} \tag{3.63}$$

is the unique monic polynomial of the smallest degree that satisfies (3.62).

A closely related notion is the error locator polynomial

$$\Lambda_e(x) \triangleq \prod_{i: e_i \neq 0} m_i(x), \tag{3.64}$$

which is of degree $\deg \Lambda_e(x) = w_D(e)$. Note that $\Lambda_e(x)$ qualifies as an error factor polynomial. In the special case where all the moduli $m_i(x), 0 \leq i \leq n - 1$, are irreducible (e.g., for irreducible polynomial remainder codes), we have

$$\gcd \left( E(x), M_n(x) \right) = \prod_{i: e_i = 0} m_i(x) \tag{3.65}$$

and thus $\Lambda_f(x) = \Lambda_e(x)$.

In any case, every error factor polynomial $\Lambda(x)$ is a multiple of $\Lambda_f(x)$. This applies, in particular, to $\Lambda_e(x)$ and thus

$$\deg \Lambda_f(x) \leq \deg \Lambda_e(x) = w_D(e). \tag{3.66}$$

The following theorem is then obvious:
Theorem 3.11 (Key Equation) The error factor polynomial satisfies
\[ A(x)M_n(x) = \Lambda_f(x)E(x) \] (3.67)
for some polynomial \( A(x) \in F[x] \) of degree smaller than \( \deg\Lambda_f(x) \). Conversely, if some monic polynomial \( G(x) \in F[x] \) satisfies
\[ A(x)M_n(x) = G(x)E(x) \] (3.68)
for some \( A(x) \in F[x] \), then \( G(x) \) is a multiple of \( \Lambda_f(x) \).

Furthermore, we have the following generalization of Theorem 2.3:

Theorem 3.12 (Alternative Key Equation) For given \( y \) and \( e \) with \( \deg\Lambda_f(x) \leq t \leq \frac{N-K}{2} \), assume that some nonzero polynomial \( \Lambda(x) \) with \( \deg\Lambda(x) \leq t \) satisfies
\[ \deg(Y(x)\Lambda(x) \mod M_n(x)) < N - t. \] (3.69)
Then \( \Lambda(x) \) is a multiple of \( \Lambda_f(x) \). Conversely, \( \Lambda(x) = \Lambda_f(x) \) is a polynomial of the smallest degree that satisfies (3.69).

For irreducible polynomial remainder codes, \( \Lambda_f(x) \) in both Theorems 3.11 and 3.12 can be replaced everywhere by \( \Lambda_e(x) \) because, in this case, \( \Lambda_f(x) = \Lambda_e(x) \).

Proof of Theorem 3.12 Note that \( Y(x) = \psi^{-1}(y) = C(x) + E(x) \) with \( \deg C(x) < K \) and \( \deg E(x) \geq N - \deg\Lambda_f(x) \geq N - t \). Consider
\[ Y(x)\Lambda(x) \mod M_n(x) = C(x)\Lambda(x) + E(x)\Lambda(x) \mod M_n(x). \] (3.70)
Under the stated assumptions, the degree of the left-hand side of (3.70) is less than \( N - t \), and also
\[ \deg \left( C(x)\Lambda(x) \right) < K + t \leq N - t. \] (3.71)
It follows that
\[ \deg \left( E(x)\Lambda(x) \mod M_n(x) \right) < N - t. \] (3.72)
Now write
\[ E(x)\Lambda(x) = Q(x)M_n(x) + \tilde{E}(x) \] (3.73)
with
\[ \tilde{E}(x) \overset{\Delta}{=} E(x)\Lambda(x) \mod M_n(x) \] (3.74)
according to the polynomial division theorem, and let

$$\bar{\Lambda}_f(x) \triangleq M_n(x)/\Lambda_f(x) = \gcd(E(x), M_n(x)).$$  \hspace{1cm} (3.75)

Since \(\deg \Lambda_f(x) \leq t\), we have \(\deg \bar{\Lambda}_f(x) \geq N - t\). Taking (3.73) modulo \(\bar{\Lambda}_f(x)\) yields

$$\tilde{E}(x) \mod \bar{\Lambda}_f(x) = 0$$  \hspace{1cm} (3.76)

since \(E(x) \mod \bar{\Lambda}_f(x) = 0\). It follows that \(\tilde{E}(x) = 0\) since \(\deg \tilde{E}(x) < N - t \leq \deg \bar{\Lambda}_f(x)\). Finally, from \(\tilde{E}(x) = 0\) and (3.74), we obtain

$$E(x)\Lambda(x) \mod M_n(x) = 0.$$  \hspace{1cm} (3.77)

But from Theorem 3.11 any nonzero polynomial \(\Lambda(x)\) that satisfies (3.77) is a multiple of the error factor polynomial (3.63). \hfill \Box

### 3.5 Error Factor-Based Interpolation

We follow the notation of Section 3.4. Let \(Y(x) = a(x) + E(x)\) denote the pre-image \(\psi^{-1}(y)\) of \(y = c + e\) with \(\psi^{-1}\) as in (3.2), where \(a(x) = \psi^{-1}(c)\) is the transmitted-message polynomial, and where \(E(x)\) denotes the pre-image \(\psi^{-1}(e)\) of the error \(e\). Note that \(\deg a(x) < K\).

The following theorem is a slight generalization of Theorem 3.8.

**Theorem 3.13 (Error Factor-based Interpolation I)**  \hspace{1cm} If \(G(x)\) is a multiple of \(\Lambda_f(x)\) with

$$\deg G(x) \leq N - K,$$  \hspace{1cm} (3.78)

then

$$a(x) = \frac{G(x)Y(x) \mod M_n(x)}{G(x)}.$$  \hspace{1cm} (3.79)

**Proof:**  \hspace{1cm} With \(Y(x) = a(x) + E(x)\) and with \(G(x)\) satisfying (3.78), we have

$$G(x)Y(x) \mod M_n(x) = G(x)(a(x) + E(x)) \mod M_n(x)$$

$$= G(x)a(x) + \tilde{E}(x)$$  \hspace{1cm} (3.80)

with

$$\tilde{E}(x) \triangleq G(x)E(x) \mod M_n(x).$$  \hspace{1cm} (3.81)

If \(G(x)\) is a multiple of \(\Lambda_f(x)\), then \(\tilde{E}(x) = 0\) by Theorem 3.11 and (3.79) follows. \hfill \Box
The following theorem is a slight generalization of Theorem 3.9.

**Theorem 3.14 (Error Factor-based Interpolation II)** If $G(x) = \gamma \Lambda_f(x)$ for some nonzero $\gamma \in F$ with

$$\text{deg } G(x) \leq N - K,$$

then

$$a(x) = Y(x) \mod M_f(x)$$

(3.83)

where $M_f(x) \triangleq M_n(x)/G(x)$.

**Proof:** Note from (3.63) that $M_f(x) = \bar{\gamma} \gcd(E(x), M_n(x))$ for some nonzero $\bar{\gamma} \in F$. Note also that $\text{deg } M_f(x) \geq K$ because $\text{deg } G(x) \leq N - K$. We then have

$$Y(x) \mod M_f(x) = (a(x) + E(x)) \mod M_f(x)$$

$$= a(x)$$

(3.84)

where the last step follows from $E(x) \mod M_f(x) = 0$. \hfill $\Box$

For irreducible polynomial remainder codes, $\Lambda_f(x)$ in both Theorems 3.13 and 3.14 can be replaced by $\Lambda_e(x)$. Theorems 3.13 and 3.14 then reduces to Theorems 3.8 and 3.9, respectively. For non-irreducible codes, however, both Theorems 3.13 and 3.14 are more general than the respective Theorems 3.8 and 3.9 because error patterns with $w_D(e) > N - K$ but $\deg \Lambda_f(x) \leq N - K$ can exist.

### 3.6 Decoding PRC via The Alternative Key Equation

Let $C$ be a polynomial remainder code of the form (3.32). For the received $y = c + e$, where $c = (c_0, \ldots, c_{n-1}) \in C$ is a transmitted codeword, and where $e = (e_0, \ldots, e_{n-1})$ is an error pattern, let $Y(x) = a(x) + E(x)$ denote the pre-image $\psi^{-1}(y)$ of $y$ with $\psi^{-1}$ as in (3.2), where $a(x) = \psi^{-1}(c)$ is the transmitted-message polynomial, and where $E(x)$ denotes the pre-image $\psi^{-1}(e)$ of the error $e$. 
3.6.1 Errors-only Decoding

It is clear from Theorem 3.12 that if \( \deg \Lambda_f(x) < \frac{N-K}{2} \), then any nonzero \( \Lambda(x) \) that satisfies

\[
\deg (Y(x)\Lambda(x) \mod M_n(x)) < (N + K)/2 \tag{3.85}
\]

must be a multiple of \( \Lambda_f(x) \). In particular, \( \Lambda(x) = \gamma \Lambda_f(x) \) (for any nonzero \( \gamma \in F \)) is a such polynomial of the smallest degree. Finding a nonzero \( \Lambda(x) = \gamma \Lambda_f(x) \) that satisfies (3.85) translates into the Partial-Inverse Problem of Section 2.1 with \( b(x) = Y(x), m(x) = M_n(x), \) and \( d = (N + K)/2 \). Once we have \( \gamma \Lambda_f(x) \), we can complete decoding by means of Theorem 3.13 or Theorem 3.14. We thus have the following decoding procedure:

1. Compute \( Y(x) = \psi^{-1}(y) \).

2. Run any algorithm of Sections 2.3 and 2.4 with \( b(x) = Y(x), m(x) = M_n(x), \) and \( d = \lceil \frac{N+K}{2} \rceil \). If \( \deg \Lambda_f(x) \leq \frac{N-K}{2} \), then the polynomial \( \Lambda(x) \) returned by the algorithm equals \( \Lambda_f(x) \) up to a scale factor.

3. Complete decoding by means of Theorem 3.13 i.e.,

\[
a(x) = \frac{\Lambda(x)Y(x) \mod M_n(x)}{\Lambda(x)} \tag{3.86}
\]

or by means of Theorem 3.14.

Note that in Step 2, because of (2.6), coefficients \( Y_\ell \) of \( Y(x) \) with

\[
\ell < \ell_{\min} \triangleq \begin{cases} 
K, & \text{if } N - K \text{ is even} \\
K + 1, & \text{if } N - K \text{ is odd} 
\end{cases} \tag{3.87}
\]

are irrelevant for finding \( \Lambda(x) \) and can be set to zero. The remaining coefficients \( Y_\ell \) are syndromes since \( a_\ell = 0 \) and \( Y_\ell = E_\ell \) for \( \ell \geq \ell_{\min} \).

3.6.2 Joint Errors-and-Erasures Decoding

In this section, we address the joint errors-and-erasures decoding of polynomial remainder codes. As in Section 2.12, let \( S_v \) with \( |S_v| \geq 1 \) be the set indexing the erasure locations. We then define the erasure locator as follows

\[
\Lambda_v(x) \triangleq \prod_{\ell \in S_v} m_\ell(x). \tag{3.88}
\]
Moreover, let
\[ \tilde{M}_n(x) \triangleq M_n(x)/\Lambda_v(x), \]  \hspace{1cm} (3.89)
which has degree
\[ \deg \tilde{M}_n(x) \triangleq \tilde{N} = N - \deg \Lambda_v(x). \]  \hspace{1cm} (3.90)

We then define
\[ \tilde{\Lambda}_f(x) \triangleq \tilde{M}_n(x)/\gcd(E(x), \tilde{M}_n(x)), \]  \hspace{1cm} (3.91)
which is a nonzero polynomial of the smallest degree that satisfies
\[ E(x)\tilde{\Lambda}_f(x) \mod \tilde{M}_n(x) = 0. \]  \hspace{1cm} (3.92)

Note that in any case, \( \tilde{\Lambda}_f(x) \) divides \( 3.63 \), and thus
\[ \deg \tilde{\Lambda}_f(x) \leq \deg \Lambda_f(x). \]  \hspace{1cm} (3.93)

Now, let
\[ \tilde{Y}(x) \triangleq Y(x) \mod \tilde{M}_n(x). \]  \hspace{1cm} (3.94)
We then have the analog of Theorem 3.12.

**Theorem 3.15**  For given \( y \) and \( e \) with \( \deg \tilde{\Lambda}_f(x) \leq t \leq \frac{\tilde{N} - K}{2} \), assume that some nonzero polynomial \( \Lambda(x) \) with \( \deg \Lambda(x) \leq t \) satisfies
\[ \deg (\tilde{Y}(x)\Lambda(x) \mod \tilde{M}_n(x)) < \tilde{N} - t. \]  \hspace{1cm} (3.95)
Then \( \Lambda(x) \) is a multiple of \( \tilde{\Lambda}_f(x) \). Conversely, \( \Lambda(x) = \tilde{\Lambda}_f(x) \) is a polynomial of the smallest degree that satisfies \( 3.95 \).

The proof is omitted since the same argument as proving Theorem 3.12 applies. We thus have the decoding procedure as in the errors-only decoding: compute \( \tilde{Y}(x) \), and run any algorithm of Sections 2.3 and 2.4 with \( b(x) = \tilde{Y}(x) \), \( m(x) = \tilde{M}_n(x) \), and \( d = \lceil \frac{\tilde{N} + K}{2} \rceil \). If \( \deg \Lambda_f(x) \leq (\tilde{N} - K)/2 \), which coincides with
\[ 2 \deg \tilde{\Lambda}_f(x) + \deg \Lambda_v(x) < N - K, \]  \hspace{1cm} (3.96)
then the polynomial returned by the algorithm will equal \( \gamma \tilde{\Lambda}_f(x) \) for some nonzero \( \gamma \in F \). We then complete decoding (by means of Theorem 3.13) by
\[ a(x) = \frac{Y(x)\Lambda(x) \mod \tilde{M}_n(x)}{\Lambda(x)} \]  \hspace{1cm} (3.97)
or by Theorem 3.14

\[ a(x) = Y(x) \mod M_f(x) \]  

(3.98)

where \( M_f(x) \triangleq \tilde{M}_n(x)/\Lambda(x) \).

Remark: when applied to decoding Reed-Solomon codes, this decoding approach reduces to the Approach-III of Section 2.12. The corresponding approaches as Approaches-I and II of Section 2.12 for polynomial remainder codes are now rather easy to develop, and thus they are omitted here. Two similar approaches can also be found in 8.

3.7 Conclusion

We studied polynomial remainder codes and their decoding more carefully than in previous work. We explicitly allowed the code symbols to be polynomials of different degrees, which leads to the two different notions of weight and distance and, correspondingly, to the two different Singleton bounds.

Our discussion of algebraic decoding revolved around the notion of an error factor polynomial, which is a generalization of an error locator polynomial. From a correct error factor polynomial, the transmitted codeword can be recovered in various ways, including the new methods for erasures-only decoding of the Chinese remainder codes.

Error factor polynomials can be found from solving the alternative key equation as in Theorem 3.12 for errors-only decoding or as in Theorem 3.15 for joint errors-and-erasures decoding. Solving such an equation translates immediately into the Partial-Inverse Problem of Section 2.1, and thus error factor polynomials can be computed by any algorithm of Sections 2.3 and 2.4. We thus have, as in Section 2.5, the partial-inverse approach to decoding polynomial remainder codes.
Chapter 4

GCD-Based Decoding of Polynomial Remainder Codes

For Reed-Solomon codes, the use of the extended gcd algorithm to compute an error locator polynomial is standard \[9,11\]. Gcd-based decoding of polynomial remainder codes was proposed by Shiozaki \[19\]. However, the assumptions in \[19\] do not cover all codes considered in Section 3.3. In particular, in \[19\], the moduli \(m_i(x)\) are assumed to have the same degree and they are implicitly assumed to be irreducible, cf. Section 4.5. In order to properly address these issues, in this chapter, gcd-based decoding is developed accordingly, and several versions of gcd-based decoding (summarized in Section 4.4) are obtained, some of which are not quite standard even when specialized to Reed-Solomon codes.

4.1 An Extended GCD Algorithm

We follow the setup of Sections 3.3 and 3.4. Let \(C\) be a polynomial remainder code of the form (3.32). For the received \(y = c + e\), where \(c = (c_0, \ldots, c_{n-1}) \in C\) is a transmitted codeword, and where \(e = (e_0, \ldots, e_{n-1})\) is an error pattern, let \(Y(x) = a(x) + E(x) = \sum_\ell Y_\ell x^\ell\) denote the pre-image \(\psi^{-1}(y)\) of \(y\) with \(\psi^{-1}\) as in (3.2), where \(a(x) = \psi^{-1}(c)\) is the transmitted-message polynomial, and where \(E(x) = \sum_\ell E_\ell x^\ell\) denotes the pre-image \(\psi^{-1}(e)\) of the error \(e\).
The general idea of gcd decoding is to compute \( \gcd(M_n(x), E(x)) \) despite the fact that \( E(x) \) is not fully known. We begin by stating the extended gcd algorithm in the following (not quite standard) form, where we assume for the moment that \( E(x) \) is fully known.

**Extended GCD Algorithm**

**Input:** :\( M_n(x) \) and \( E(x) \) with \( \deg M_n(x) > \deg E(x) \).

**Output:** polynomials \( \tilde{r}(x), s(x), t(x) \in F[x] \) where \( s(x) \) and \( t(x) \) satisfy \( s(x) \cdot M_n(x) + t(x) \cdot E(x) = 0 \), and where \( \tilde{r}(x) = \gamma \gcd(M_n(x), E(x)) \) for some nonzero \( \gamma \in F \).

1. if \( E(x) = 0 \) begin
2. \( \tilde{r}(x) := M_n(x) \), \( s(x) := 0 \), \( t(x) := 1 \)
3. return \( \tilde{r}(x), s(x), t(x) \)
4. end
5. \( r(x) := M_n(x) \)
6. \( \tilde{r}(x) := E(x) \)
7. \( s(x) := 1 \)
8. \( t(x) := 0 \)
9. \( \tilde{s}(x) := 0 \)
10. \( \tilde{t}(x) := 1 \)
11. loop begin
12. \( i := \deg r(x) \)
13. \( j := \deg \tilde{r}(x) \)
14. while \( i \geq j \) begin
15. \( q(x) := \frac{r_i}{\tilde{r}_j} x^{i-j} \)
16. \( r(x) := r(x) - q(x) \cdot \tilde{r}(x) \)
17. \( s(x) := s(x) - q(x) \cdot \tilde{s}(x) \)
18. \( t(x) := t(x) - q(x) \cdot \tilde{t}(x) \)
19. \( i := \deg r(x) \)
20. end
21. if \( r(x) = 0 \) begin
22. return \( \tilde{r}(x), s(x), t(x) \)
23. end
24. \( (r(x), \tilde{r}(x)) := (\tilde{r}(x), r(x)) \)
25. \( (s(x), \tilde{s}(x)) := (\tilde{s}(x), s(x)) \)
26. \( (t(x), \tilde{t}(x)) := (\tilde{t}(x), t(x)) \)
27. end

The inner loop between lines 14 and 20 essentially computes the division of \( r(x) \) by \( \tilde{r}(x) \). In line 15, \( r_i \) denotes the coefficient of \( x^i \) in \( r(x) \) and \( \tilde{r}_j \)
4.1 An Extended GCD Algorithm

denotes the coefficient of \(x^j\) in \(\tilde{r}(x)\). For polynomials over \(F = \text{GF}(2)\), the scalar division \(r_i/\tilde{r}_j\) in line 15 disappears.

**Theorem 4.1 (GCD Loop Invariants)** The condition

\[
gcd \left( M_n(x), E(x) \right) = gcd \left( r(x), \tilde{r}(x) \right)
\]

(4.1)

holds everywhere after line 6. The condition

\[
r(x) = s(x) \cdot M_n(x) + t(x) \cdot E(x)
\]

(4.2)

holds both between lines 13 and 14 and between lines 20 and 21. The condition

\[
\deg M_n(x) = \deg \tilde{r}(x) + \deg t(x)
\]

(4.3)

holds between lines 20 and 21.

Equations (4.1) and (4.2) are the standard loop invariants of extended gcd algorithms, cf. e.g. [11]. The proof of Theorem 4.1 is given in Section 4.8.

**Theorem 4.2 (GCD Output)** When the algorithm terminates, we have both

\[
\tilde{r}(x) = \gamma \gcd \left( M_n(x), E(x) \right)
\]

(4.4)

\[
= \gamma \frac{M_n(x)}{\Lambda_f(x)}
\]

(4.5)

for some nonzero \(\gamma \in F\) and

\[
t(x) = \tilde{\gamma} \Lambda_f(x)
\]

(4.6)

for some nonzero \(\tilde{\gamma} \in F\). Moreover, the returned \(s(x)\) and \(t(x)\) satisfy

\[
s(x) \cdot M_n(x) + t(x) \cdot E(x) = 0.
\]

(4.7)

**Proof:** If \(E(x) = 0\), the algorithm terminates at line 3 and (4.4)–(4.7) are easily verified.

We now prove the case where \(E(x) \neq 0\). Equation (4.4) follows from (4.1) and (4.5) follows from (3.63). It remains to prove (4.6) and (4.7). With \(r(x) = 0\) and from (4.2), Equation (4.7) follows. We then conclude from the second part of Theorem 3.11 that \(t(x)\) is a multiple of \(\Lambda_f(x)\). Finally, it follows from (4.3) and (4.5) that \(t(x)\) and \(\Lambda_f(x)\) have the same degree. \(\square\)
From (4.6), we see that the gcd algorithm computes the error factor polynomial \( \Lambda_f \) (up to a scale factor). The main idea of gcd decoding (discovered by Sugiyama [9]) is that this still works even if \( E(x) \) is only partially known.

### 4.2 Modifications for Partially Known \( E(x) \)

Recall from (3.38) and (3.39) that \( N \triangleq \deg M_n(x) \) and \( K \triangleq \deg M_k(x) \); \( K \) is the dimension of the code as a subspace of \( F^N \). Since \( Y(x) = a(x) + E(x) \) and \( \deg a(x) < K \), the receiver knows the coefficients \( E_K, E_{K+1}, \ldots, E_{N-1} \) of \( E(x) \) (from \( Y_K, Y_{K+1}, \ldots, Y_{N-1} \) of \( Y(x) \)), but not \( E_0, \ldots, E_{K-1} \). With the following modifications, the Extended GCD Algorithm of Section 4.1 can still be used to compute (4.6).

**Partial GCD Algorithm I**

**Input:** \( M_n(x) \) and \( Y(x) \) with \( \deg M_n(x) > \deg Y(x) \).

**Output:** \( r(x), s(x) \) and \( t(x) \), cf. Theorem 4.3 below.

The algorithm is the same as the Extended GCD Algorithm of Section 4.1 except for the following changes:

- **Line 1** if \( \deg Y(x) < K \) begin
- **Line 2** \( r(x) := Y(x), s(x) := 0, t(x) := 1 \)
- **Line 6** \( \bar{r}(x) := Y(x) \)
- **Line 21**

\[
\text{if } \deg r(x) < \deg t(x) + K \text{ begin} \tag{4.8}
\]

or alternatively

\[
\text{if } \deg r(x) < (N + K)/2 \text{ begin} \tag{4.9}
\]

**Theorem 4.3** If

\[
\deg \Lambda_f(x) \leq (N - K)/2, \tag{4.10}
\]

then the Partial GCD Algorithm I (with either (4.8) or (4.9)) returns the same polynomials \( s(x) \) and \( t(x) \) (after the same number of iterations) as the Extended GCD Algorithm of Section 4.1. Moreover, the returned \( r(x) \) is such that

\[
r(x) = t(x)a(x). \tag{4.11}
\]

The proof is given in Section 4.8. Note that \( a(x) \) can be recovered directly from (4.11).
4.3 Alternative Modifications for Partially Known \( E(x) \)

The Partial GCD Algorithm I of the previous section involves a lot of computations with the unknown lower parts of \( E(x) \). These computations are avoided in the following algorithm, which works only with the known part of \( E(x) \) as follows. Let

\[
E_U(x) \triangleq \sum_{\ell=0}^{N-K-1} E_{K+\ell} x^\ell = \sum_{\ell=0}^{N-K-1} Y_{K+\ell} x^\ell, \tag{4.12}
\]

which is the known upper part of \( E(x) = \sum_{\ell=0}^{N-1} E_\ell x^\ell \), and let

\[
M_U(x) \triangleq \sum_{\ell=0}^{N-K} (M_n)_{K+\ell} x^\ell \tag{4.13}
\]

be the corresponding upper part of \( M_n(x) = \sum_{\ell=0}^{N} (M_n)_{\ell} x^\ell \).

**Partial GCD Algorithm II**

Input: \( M_U(x) \) and \( E_U(x) \) with \( \deg M_U(x) > \deg E_U(x) \).

Output: \( s(x) \) and \( t(x) \), cf. Theorem 4.4 below.

The algorithm is the same as the Extended GCD Algorithm of Section 4.1 except for the following changes:

- Line 1: if \( E_U(x) = 0 \) begin
- Line 2: \( s(x) := 0, \ t(x) := 1 \)
- Line 5: \( r(x) := M_U(x) \)
- Line 6: \( \tilde{r}(x) := E_U(x) \)
- Line 21:
  \[
  \text{if } \deg r(x) < \deg t(x) \text{ begin} \tag{4.14}
  \]
  or alternatively
  \[
  \text{if } \deg r(x) < (N - K)/2 \text{ begin} \tag{4.15}
  \]

\[\square\]
Theorem 4.4  If the condition (4.10) is satisfied, then the Partial GCD Algorithm II (with either (4.14) or (4.15)) returns the same polynomials \( s(x) \) and \( t(x) \) (after the same number of iterations) as the Extended GCD Algorithm of Section 4.4.

The proof is given in Section 4.9. Note, however, that this algorithm does not compute \( r(x) \) as in (4.11).

4.4 Summary of Decoding

We can now put together several decoding algorithms that consist of the following three steps. The relation of all these decoding algorithms to the prior literature is discussed in Section 4.5.

1. **Transform**: Compute \( Y(x) = \psi^{-1}(y) \). If \( \deg Y(x) < K \), we conclude \( E(x) = 0 \) and \( a(x) = Y(x) \), and the following two steps can be skipped.

2. **Partial GCD**: If \( \deg Y(x) \geq K \), run either the Partial GCD Algorithm I (Section 4.2) or the Partial GCD Algorithm II (Section 4.3). Either algorithm yields the polynomial \( t(x) = \tilde{\gamma}\Lambda_f(x) \) (for some scalar \( \tilde{\gamma} \in F \)) provided that \( \deg \Lambda_f(x) \leq (N - K)/2 \). If \( \deg t(x) > (N - K)/2 \), we declare a decoding failure.

   Depending on Step 3 (below), the computation of the polynomials \( s(x) \) and \( \tilde{s}(x) \) may be unnecessary. In this case, lines 7, 9, 17, and 25 of the gcd algorithm can be deleted.

3. **Recovery**: Recover \( a(x) \) by any of the following methods:

   (a) From (3.79), we have

\[
a(x) = \frac{t(x)Y(x) \mod M_n(x)}{t(x)} \tag{4.16}
\]

   (If the numerator of (4.16) is not a multiple of \( t(x) \) or if \( \deg a(x) \geq K \), then decoding failed due to some uncorrectable error.)

   (b) From (3.83), we obtain

\[
a(x) = Y(x) \mod M_{\tilde{f}}(x) \tag{4.17}
\]
where $M_f(x) \triangleq M_n(x)/t(x)$.

(If $M_n(x)$ is not a multiple of $t(x)$ or if $\deg a(x) \geq K$, then decoding failed due to some uncorrectable error.)

(c) When using the Partial GCD Algorithm I in the Step 2, we can compute $a(x) = r(x)/t(x)$ according to (4.11). (If $t(x)$ does not divide $r(x)$ or if $\deg a(x) \geq K$, we declare a decoding failure.)

(d) Alternatively, from (4.7), we can compute

$$E(x) = \frac{-s(x) \cdot M_n(x)}{t(x)}$$

and then obtain $a(x) = Y(x) - E(x)$. (If the numerator of (4.18) is not a multiple of $t(x)$ or if $\deg a(x) \geq K$, we declare a decoding failure.)

The computation can be simplified as follows. Let $E_L(x) \triangleq E(x) - x^K E_U(x)$ denote the unknown part of $E(x)$. Then

$$E_L(x) = \frac{-s(x) \cdot M_n(x) - x^K t(x) E_U(x)}{t(x)}$$

and $a(x)$ can be recovered by $a(x) = \sum_{\ell=0}^{K-1} Y_\ell x^{\ell} - E_L(x)$.

As stated, the described decoding algorithms are guaranteed to correct all errors $e$ with $\deg \Lambda_f(x) \leq t_D$, which by (3.66) implies that they also correct all errors $e$ with $w_D(e) \leq t_D$ (3.59). If the code satisfies the Ordered-Degree Condition (3.33) as well as the additional condition

$$\deg m_k(x) = \cdots = \deg m_{n-1}(x),$$

then the algorithm is guaranteed to correct also all errors $e$ with $w_H(e) \leq t_H$ (3.60) since in this case, from (3.64), $w_H(e) \leq t_H$ implies $w_D(e) \leq t_D$.

### 4.5 Relation to Prior Work

The idea of gcd-based decoding is due to Sugiyama [9] and its application to polynomial remainder codes is due to Shiozaki [19]. As it turns out, most (and perhaps all) gcd-based decoding algorithms in the literature, both for Reed-Solomon codes and for polynomial residue codes,
are essentially identical to one of the algorithms of Section 4.4. However, even when specialized to Reed-Solomon codes, no single paper (not even \[35,36\]) seems to cover all these algorithms. In particular, recovering \(a(x)\) by (4.16) or by (4.17) does not seem to have appeared in the literature. For Reed-Solomon codes, the work by Gao \[20\] appears to be the most pertinent, see also \[35,36\]. As for polynomial remainder codes, our algorithms overcome the limitations of Shiozaki’s algorithm \[19\] as will be discussed below.

Relation to Gao’s Decoding Algorithms

In the same paper \[20\] from 2003, Gao proposed two algorithms for decoding Reed-Solomon codes. Each algorithm comprises three steps, and the first step of each algorithm is essentially Step 1 (“Transform”) of Section 4.4.

Gao’s first algorithm: Step 2 of this algorithm is essentially the Partial GCD Algorithm I of Section 4.2 with (4.9) as the stopping condition. Step 3 is identical to Step 3.c in Section 4.4.

As pointed out in \[36\], this algorithm is actually identical to Shiozaki’s 1988 algorithm for decoding Reed-Solomon codes \[19\].

Gao’s second algorithm: The stopping condition of the gcd-algorithm (Step 2) as stated in \[20\] is not quite correct: it should be changed from \(\text{deg} \ g(x) < (d+1)/2\) to \(\text{deg} \ g(x) < (d−1)/2\) where \(d \triangleq n−k+1\) is the minimum Hamming distance of the code.

With this correction, Step 2 of this algorithm is identical to the Partial GCD Algorithm II of Section 4.2 with (4.15) as the stopping condition. Step 3 of the algorithm turns out to be equivalent to the first part of 3.d in Section 4.4 i.e., computing \(a(x) = Y(x) − E(x)\) with \(E(x)\) as in (4.18).

Relation to Shiozaki’s Decoding Algorithms

In \[19\], Shiozaki proposed a new version of gcd-based decoding for Reed-Solomon codes, which he also extended to polynomial remainder codes. (For Reed-Solomon codes, Shiozaki’s algorithm is equivalent to Gao’s first decoding algorithm, as noted above.) Shiozaki’s algorithm also consists of three steps: the first step agrees with Step 1 in Section 4.4, the second step is equivalent to the Partial
GCD Algorithm I with (4.9) as the stopping condition, and the third step is identical to Step 3.c of Section 4.4.

However, the assumptions in [19] do not cover all codes considered in the present paper. First, it is assumed in [19] that all the moduli \( m_i(x), 0 \leq i \leq n - 1 \), have the same degree.

Second, the argument given in [19] seems to assume that all the moduli are irreducible although this assumption is not stated explicitly. Specifically, Shiozaki derived a congruence (see (37) in [19]) involving an error locator polynomial as defined in (3.64), and then used the gcd-based decoding algorithm to solve the congruence. However, if the moduli are not irreducible, then the gcd-based decoding algorithm will find an error factor polynomial (3.63) (as shown in our Theorems 4.2 and 4.3) rather than an error locator polynomial.

4.6 Conclusion

We have investigated the gcd-based decoding of polynomial remainder codes more carefully than in previous work. We explicitly allowed the code symbols to be polynomials of different degrees, and did not require the moduli to be irreducible. We showed that error factor polynomials can be computed by a suitably adapted partial gcd algorithm. From a correct error factor polynomial, the transmitted codeword can be recovered in various ways. We then obtained several versions of such decoding algorithms, which generalize previous work and which include the published gcd-based decoders of Reed-Solomon codes as special cases. The gcd-based decoding also complements the previous chapter on decoding polynomial remainder codes.

4.7 An Extension

This section simply serves as a supplement to Section 4.4. Assume that the code satisfies the Ordered-Degree Condition (3.33) but not the additional condition (4.20). In this case, we can still correct all errors \( e \) with \( w_H(e) \leq t_H \) (in addition to all errors with \( w_D(e) \leq t_D \)) by the procedure stated below, which, however, is practical only in special cases.

The procedure needs the following Theorem 4.5. Recall that \( C \) is a code of the form (3.32), and recall from (3.60) that \( t_H \triangleq \left\lfloor \frac{n-k}{2} \right\rfloor \). Now, let \( N_{\text{zero}}(G) \) denote the number of indices \( j \in \{0, \ldots, n-1\} \) such that \( G(x) \mod m_j(x) = 0 \). Note that \( N_{\text{zero}}(A_e) = w_H(e) \). If the code
Let $C$ further satisfies the Ordered-Degree Condition \((3.33)\), we have the following theorem.

**Theorem 4.5 (Error Locator Test)** Let $C$ be a polynomial remainder code that satisfies the Ordered-Degree Condition and let $y = \psi(a) + e$ as above. For some set $S \subset \{0, 1, \ldots, n - 1\}$ of indices, let $G(x) = \prod_{i \in S} m_i(x) \neq 0$ and let

$$Z(x) \triangleq G(x)Y(x) \mod M_n(x).$$

Assume that the following conditions are satisfied:

1. $w_H(e) \leq t_H$
2. $N_{\text{zero}}(G) \leq t_H \text{ and } \deg G(x) \leq \sum_{i=n-t_H}^{n-1} \deg m_i(x)$
3. $G(x)$ divides $Z(x)$
4. $\deg Z(x) - \deg G(x) < K$.

Then, $G(x)$ is a multiple of $\Lambda_e(x)$ and $Z(x) = G(x)a(x)$.

Note that the conditions in the theorem are satisfied for $G(x) = \Lambda_e(x)$. For the proof of Theorem 4.5, we refer to Section 4.2 of [7].

We thus have the following procedure.

**Decoder with List of Special Error Positions**

First, run the gcd decoder of the previous section. If it succeeds, stop. Otherwise, let $S_{\Lambda}$ be a precomputed list of candidate error locator polynomials $G(x)$ with $N_{\text{zero}}(G) \leq t_H$ and $\deg G(x) > (N - K)/2$. Check if any $G(x) \in S_{\Lambda}$ satisfies all conditions of Theorem 4.5. If such a polynomial $G(x)$ exists, we conclude that it is a multiple of the error locator polynomial and we compute $a(x)$ from \((3.79)\) or \((3.83)\).

\[\square\]

### 4.8 Proof of Theorem 4.3

In this section, we first prove the loop invariant properties of the Extended GCD Algorithm in Section 4.1 and the Partial GCD Algorithm I in Section 4.2, and then proceed to prove Theorem 4.3.

We begin with the Extended GCD Algorithm of Section 4.1. In order to prove Theorem 4.1, we first recall that, for $R = \mathbb{Z}$ or $R = F[x]$ for some field $F$,

$$\gcd(a, b) = \gcd(a + qb, b) \quad (4.21)$$
for all $a, b, q \in R$, provided that $a$ and $b$ are not both zero. It follows that (4.1) holds everywhere after line 6.

The other claims of Theorem 4.1 are covered by the following lemma.

**Lemma 4.1 (GCD Loop Invariant)**  For the Extended GCD Algorithm in Section 4.1, the condition

$$ r(x) = s(x) \cdot M_n(x) + t(x) \cdot E(x) $$ (4.22)

holds both between lines 13 and 14 and between lines 20 and 21. For the Partial GCD Algorithm I in Section 4.2, the condition

$$ r(x) = s(x) \cdot M_n(x) + t(x) \cdot Y(x) $$ (4.23)

also holds both between lines 13 and 14 and between lines 20 and 21.

For both algorithms, the conditions

$$ \text{deg } r(x) < \text{deg } \tilde{r}(x) $$ (4.24)
$$ \text{deg } t(x) > \text{deg } \tilde{t}(x) $$ (4.25)
$$ \text{deg } M_n(x) = \text{deg } \tilde{r}(x) + \text{deg } t(x) $$ (4.26)

hold between lines 20 and 21.

Specifically, let $\delta_\ell$ denote the degree of $q(x)$ (line 15) in the first iteration of the while block (lines 14-20) of the $\ell$-th loop iteration. Then, for the respective algorithms,

$$ \text{deg } t(x) = \text{deg } \tilde{t}(x) + \delta_\ell = \sum_{v=1}^{\ell} \delta_v $$ (4.27)

holds between lines 20 and 21 in the $\ell$-th loop iteration.

**Proof:** Conditions (4.22) and (4.23) are loop invariants (of the respective algorithms), as is easily verified. Inequality (4.24) is obvious. It remains to prove (4.25)–(4.27). For both algorithms, assume the conditions

$$ \text{deg } r(x) > \text{deg } \tilde{r}(x) $$ (4.28)
$$ \text{deg } t(x) < \text{deg } \tilde{t}(x) $$ (4.29)
$$ \text{deg } M_n(x) = \text{deg } r(x) + \text{deg } \tilde{t}(x) $$ (4.30)

hold between lines 13 and 14 in the $\ell$-th loop iteration. Note that $r(x)$, $\tilde{r}(x)$, $t(x)$, and $\tilde{t}(x)$ are initialized to $M_n(x)$, $E(x)$ or $Y(x)$, 0, and 1,
respectively; thus (4.28)–(4.30) obviously hold between lines 13 and 14 in the first iteration. In the following, we begin with \( \ell = 1 \) and then complete the proof by induction.

For both algorithms, let \( d_\ell = \deg r(x) \) denote the degree of \( r(x) \) between lines 13 and 14 in the \( \ell \)-th loop iteration, and let \( \delta_\ell \) denote the degree of \( q(x) \) (line 15) in the first iteration of the while block (lines 14–20) of the \( \ell \)-th loop iteration. Note that \( \delta_\ell = d_\ell - \deg \tilde{r}(x) > 0 \) and from (4.30)

\[
\deg M_n(x) = d_\ell + \deg \tilde{t}(x) \tag{4.31}
\]

Recall that, from (4.29), \( \deg t(x) < \deg \tilde{t}(x) \) holds before entering the while block, and recall the update rule for \( t(x) \) in line 18. Clearly, in the first execution of line 18 the degree of \( t(x) \) is increased to \( \deg \tilde{t}(x) + \delta_\ell \), and further iterations inside the while block will not change \( \deg t(x) \) since \( \deg q(x) \) decreases in each iteration. It follows that \( \deg t(x) = \deg \tilde{t}(x) + \delta_\ell \) holds between lines 20 and 21 and in particular, \( \deg t(x) = \delta_1 \) holds when \( \ell = 1 \) because \( \deg \tilde{t}(x) = 0 \) holds throughout the while block of the first loop iteration. Thus, (4.25) and (4.27) both hold between lines 20 and 21 in the first loop iteration. Further, since \( \delta_\ell = d_\ell - \deg \tilde{r}(x) \), we have

\[
\deg t(x) = \deg \tilde{t}(x) + d_\ell - \deg \tilde{r}(x) \tag{4.32}
\]

\[
= \deg M_n(x) - \deg \tilde{r}(x), \tag{4.33}
\]

where the last step follows from (4.31), and thus (4.26) holds between lines 20 and 21 in the \( \ell \)-th loop iteration.

After the swaps of the corresponding auxiliary polynomials in lines 24–26, the conditions (4.28)–(4.30) hold again between lines 13 and 14 for the subsequent loop iteration. In particular, for \( \ell = 2 \), \( \deg t(x) = \delta_1 \) holds between lines 13 and 14 in the second loop iteration. The proof is then completed by induction. \( \square \)

We now start to prove Theorem 4.3. If \( E(x) = 0 \), which implies \( \deg Y(x) < K \), Theorem 4.3 holds obviously; we thus prove in the following only the case where \( E(x) \neq 0 \). For the Partial GCD Algorithm I in Section 4.2 let \( g \) denote the largest integer such that the coefficient of \( x^g \) of either \( r(x) \) or of \( \tilde{r}(x) \) is unknown, or alternatively let \( g \) denote the largest integer such that the coefficient of \( x^g \) of either \( r(x) \) or of \( \tilde{r}(x) \) is “probably unmatched” with the corresponding \( r(x) \) or the corresponding \( \tilde{r}(x) \) in the Extended GCD Algorithm of Section 4.1 when we run both algorithms simultaneously. Clearly, the algorithm starts with
\[ g = K - 1, \text{ since the coefficients } E_0, E_1, \ldots, E_{K-1} \text{ of } \tilde{r}(x) := Y(x) \text{ (line 6) are unknown. Moreover, let } h \triangleq \max \{ \deg r(x), \deg \tilde{r}(x) \}. \] Clearly, the algorithm starts with \( h = \deg M_n(x) = N. \)

**Lemma 4.2** For the Partial GCD Algorithm I of Section 4.2, let \( \delta_\ell \) denote the degree of \( q(x) \) in the first iteration of the while block (lines 14–20) of the \( \ell \)-th loop iteration. If \( h - g > 2 \delta_1 \) holds between lines 13 and 14, then the value of \( q(x) \) (line 15) throughout the while block in the \( \ell \)-th loop iteration is exactly the same as the corresponding one of the Extended GCD Algorithm of Section 4.1 in the same loop iteration. In addition, \( g = (K - 1) + \sum_{v=1}^{\ell} \delta_v \) and \( h = N - \sum_{v=1}^{\ell} \delta_v \) both hold between lines 20 and 21 in the \( \ell \)-th loop iteration.

**Proof:** We will prove this theorem by induction. Recall that the update rule for \( r(x) \) in line 16 is
\[
r(x) := r(x) - q(x) \cdot \tilde{r}(x). \tag{4.34}
\]
In the first loop iteration, \( h = \deg r(x) = N \) and \( g = K - 1 \) clearly hold between lines 13 and 14, and \( g \) is the largest integer such that the coefficient of \( x^g \) of \( \tilde{r}(x) \) is unknown. If \( h - g > 2 \delta_1 \) holds between lines 13 and 14, then the first execution of (4.34) in the while block increases \( g \) by \( \delta_1 \); afterwards, further iterations in the same block will not change \( g \) since \( \deg q(x) \) decreases in each iteration. Moreover, after executing the while block, \( h = \deg \tilde{r}(x) = N - \delta_1 \) holds between lines 20 and 21.

It is also easily seen that throughout the while block, the value of \( q(x) \) in line 15 is exactly identical to the corresponding one of the Extended GCD Algorithm.

Note that the increased \( g \), i.e., after the first execution of (4.34), will become to denote the largest integer such that the coefficient of \( x^g \) of \( r(x) \) is unknown. It follows after the swap of \( r(x) \) and \( \tilde{r}(x) \) in line 24 that the increased \( g \) will again become to denote the largest integer such that the coefficient of \( x^g \) of \( \tilde{r}(x) \) is unknown between lines 13 and 14 for subsequent loop iteration, and the decreased \( h \) will again become to denote \( \deg r(x) \) between lines 13 and 14 for subsequent loop iteration. The proof is then completed by induction. \( \Box \)

Since \( h - g = N - K + 1 \) holds between lines 13 and 14 in the first loop iteration, it follows from Lemma 4.2 that if
\[
2 \sum_{v=1}^{\ell} \delta_v < N - K + 1, \tag{4.35}
\]
then, from the first to the $\ell$-th loop iteration, $q(x)$ and thus $s(x)$ and $t(x)$ are exactly the same as in the Extended GCD Algorithm. Moreover from Lemma 4.1, $\deg t(x) = \sum_{v=1}^{\ell} \delta_v$ holds between lines 20 and 21. In order to obtain (4.6), which implies that $\deg t(x) = \deg \Lambda_f(x)$, it turns out from (4.35) that if

$$2 \deg \Lambda_f(x) \leq N - K,$$

which agrees with (4.10), then the algorithm maintains exactly the same $s(x)$ and $t(x)$ as the Extended GCD Algorithm of Section 4.1 until $\deg t(x) = \deg \Lambda_f(x)$.

It remains to argue the validity of (4.8) and (4.9) (i.e., line 21 in the Partial GCD Algorithm I) as appropriate terminating conditions. Assume now that (4.10) is satisfied and suppose the Extended GCD Algorithm (in Section 4.1) terminates (at line 22) in the $\mu$-th loop iteration. We will show in the following that the Partial GCD Algorithm I also terminates (at line 22) in the $\mu$-th loop iteration.

As shown above, since both the gcd algorithms maintain exactly the same $s(x)$ and $t(x)$ until $\deg t(x) = \deg \Lambda_f(x)$, clearly, before the $\mu$-th loop iteration,

$$\deg t(x) < \deg \Lambda_f(x) \leq (N - K)/2$$

holds between lines 20 and 21; moreover, by (4.26) of Lemma 4.1,

$$\deg \tilde{r}(x) = \deg M_n(x) - \deg t(x)$$

holds between lines 20 and 21. Further, from (4.25), $\deg t(x) > \deg \tilde{t}(x)$ holds as well between lines 20 and 21. Therefore,

$$\deg \tilde{r}(x) > (N + K)/2 > \deg t(x) + K > \deg \tilde{t}(x) + K$$

holds between lines 20 and 21 in every but before the $\mu$-th loop iteration. It then follows after swapping all auxiliary polynomials in lines 24–26 that

$$\deg r(x) > (N + K)/2 > \deg \tilde{t}(x) + K > \deg t(x) + K$$

holds between lines 13 and 14 for each subsequent loop iteration. Then, after executing the while block in the $\mu$-th loop iteration, the Extended
GCD Algorithm in Section 4.1 terminates with \( r(x) = 0 \), and (4.7) holds; meanwhile, for the Partial GCD Algorithm I, we obtain the desired \( t(x) \) (with \( \deg t(x) = \deg \Lambda_f(x) \)) and \( s(x) \), and we have from (4.23)

\[
\begin{align*}
  r(x) &= s(x)M_n(x) + t(x)Y(x) \quad (4.43) \\
         &= s(x)M_n(x) + t(x)E(x) + t(x)a(x) \quad (4.44) \\
         &= t(x)a(x) \quad (4.45)
\end{align*}
\]

of \( \deg r(x) = \deg t(x) + \deg a(x) < \deg t(x) + K \), where (4.44) to (4.45) follows from (4.7). Finally, since from (4.42) \( \deg r(x) > \deg t(x) + K \) holds between lines 13 and 14 but from (4.45) \( \deg r(x) < \deg t(x) + K \) holds between lines 20 and 21, thus the correctness of (4.8) as a terminating condition is guaranteed; meanwhile from (4.45) we obtain (4.11). As for (4.9), since from (4.42) \( \deg r(x) > (N + K)/2 \) holds between lines 13 and 14 but (from (4.45) and then (4.10)) \( \deg r(x) < \deg t(x) + K = \deg \Lambda_f(x) + K \leq (N + K)/2 \) holds between lines 20 and 21, we thus conclude that (4.9) can serve as an alternative terminating condition.

4.9 Proof of Theorem 4.4

In this section, we prove Theorem 4.4 in an analogous way as proving Theorem 4.3. The following theorem is an analog of Lemma 4.1.

Lemma 4.3 (GCD Loop Invariant)  For the Partial GCD Algorithm II in Section 4.3, the condition

\[
\begin{align*}
  r(x) &= s(x) \cdot M_U(x) + t(x) \cdot E_U(x) \quad (4.46)
\end{align*}
\]

holds both between lines 13 and 14 and between lines 20 and 21; moreover, the conditions

\[
\begin{align*}
  \deg r(x) &< \deg \tilde{r}(x) \quad (4.47) \\
  \deg t(x) &> \deg \tilde{t}(x) \quad (4.48) \\
  \deg M_U(x) &= \deg \tilde{r}(x) + \deg t(x) \quad (4.49)
\end{align*}
\]

hold between lines 20 and 21.

Specifically, let \( \delta_\ell \) denote the degree of \( q(x) \) (line 15) in the first iteration of the while block (lines 14–20) of the \( \ell \)-th loop iteration. Then, \( \deg t(x) = \deg \tilde{t}(x) + \delta_\ell = \sum_{v=1}^{\ell} \delta_v \) holds between lines 20 and 21 in the \( \ell \)-th loop iteration.
The proof of Lemma 4.3 is the same as the proof of Lemma 4.1 except for replacing the $M_n(x)$ in the proof of Lemma 4.1 by $M_U(x)$, and is thus omitted.

We now start to prove Theorem 4.4. If $E(x) = 0$, which implies $E_U(x) = 0$, Theorem 4.4 holds obviously; we thus prove in the following only the case where $E(x) \neq 0$. For the Partial GCD Algorithm II of Section 4.3, let $g$ denote the largest integer such that $x^g$ of either $r(x)$ or of $\tilde{r}(x)$ is unknown. Clearly, with $M_U(x)$ and $E_U(x)$ as inputs, the algorithm starts with $g = -1$. Moreover, let $h = \max\{\deg r(x), \deg \tilde{r}(x)\}$. Clearly, the algorithm starts with $h = \deg M_U(x) = N - K$.

**Lemma 4.4** For the Partial GCD Algorithm II in Section 4.3, let $\delta_\ell$ denote the degree of $q(x)$ in the first iteration of the while block (lines 14–20) of the $\ell$-th loop iteration. If $h - g > 2\delta_\ell$ holds between lines 13 and 14 then the value of $q(x)$ (line 15) throughout the while block in the $\ell$-th loop iteration is exactly the same as the corresponding one of the Extended GCD Algorithm of Section 4.1 in the same loop iteration. In addition, $g = -1 + \sum_{\ell=1}^{\ell} \delta_v$ and $h = N - K - \sum_{\ell=1}^{\ell} \delta_v$ both hold between lines 20 and 21 in the $\ell$-th loop iteration.

The proof is similar to that of Lemma 4.2 and is thus omitted. Since $h - g = N - K + 1$ holds between lines 13 and 14 in the first loop iteration, it follows from Lemma 4.4 that if $2 \sum_{\ell=1}^{\ell} \delta_v < N - K + 1$, then, from the first to the $\ell$-th loop iteration, $q(x)$ and thus $s(x)$ and $t(x)$ are exactly the same as in the Extended GCD Algorithm. Moreover, from Lemma 4.3 $\deg t(x) = \sum_{\ell=1}^{\ell} \delta_v$ holds between lines 20 and 21. In order to obtain (4.6), which implies that $\deg t(x) = \deg \Lambda_f(x)$, it turns out that if

$$2 \deg \Lambda_f(x) \leq N - K,$$

which agrees with (4.10), then the algorithm maintains exactly the same $s(x)$ and $t(x)$ as the Extended GCD Algorithm of Section 4.1 until $\deg t(x) = \deg \Lambda_f(x)$.

It remains to argue the validity of (4.14) and (4.15) as appropriate terminating conditions. Assume that (4.14) is satisfied and suppose the Extended GCD Algorithm (in Section 4.1) terminates (at line 22) in the $\mu$-th loop iteration. As shown above, it has been clear that the Extended GCD Algorithm in Section 4.1 and the Partial GCD Algorithm II maintain exactly the same $s(x)$ and $t(x)$ until $\deg t(x) = \deg \Lambda_f(x)$. Thus, before the $\mu$-th loop iteration

$$\deg t(x) < \deg \Lambda_f(x) \leq (N - K)/2$$

(4.51)
holds between lines 20 and 21; moreover, by (4.49) of Lemma 4.3,

$$\deg \tilde{r}(x) = \deg M_U(x) - \deg t(x)$$  \hspace{1cm} (4.52)

$$> (N - K)/2$$  \hspace{1cm} (4.53)

$$> \deg t(x)$$  \hspace{1cm} (4.54)

also holds between lines 20 and 21 for the Partial GCD Algorithm II. Further, from (4.48), \(\deg t(x) > \deg \tilde{t}(x)\) holds as well between lines 20 and 21. Therefore, for the Partial GCD Algorithm II,

$$\deg \tilde{r}(x) > (N - K)/2 > \deg t(x) > \deg \tilde{t}(x)$$  \hspace{1cm} (4.55)

holds between lines 20 and 21 in every but before the \(\mu\)-th loop iteration. It then follows after swapping all auxiliary polynomials in lines 24–26 that

$$\deg r(x) > (N - K)/2 > \deg \tilde{t}(x) > \deg t(x)$$  \hspace{1cm} (4.56)

holds between lines 13 and 14 for each subsequent loop iteration. Then, after executing the while block in the \(\mu\)-th loop iteration, we obtain the desired \(t(x)\) (with \(\deg t(x) = \deg \Lambda_f(x)\)) and \(s(x)\) that coincide with the corresponding ones of the Extended GCD Algorithm in Section 4.1; thus \(t(x)\) and \(s(x)\) (in the Partial GCD Algorithm II) at this moment satisfy both (4.46) and (4.7). From (4.7), we have

$$-s(x)M_n(x) = t(x)E(x)$$  \hspace{1cm} (4.57)

with \(\deg s(x) < \deg t(x)\). Note that (4.57) can also be written as

$$-s(x)(x^K M_U(x) + M_L(x)) = t(x)(x^K E_U(x) + E_L(x)),$$  \hspace{1cm} (4.58)

where \(M_U(x)\) and \(E_U(x)\) are defined in Section 4.3 and \(M_L(x) = M_n(x) - x^K M_U(x)\) and \(E_L(x) = E(x) - x^K E_U(x)\). Further, let

$$V(x) \triangleq -s(x)M_L(x) - t(x)E_L(x) = \sum_{\ell=0} V_\ell x^\ell,$$

which is of degree \(\deg V(x) \leq (K - 1) + \deg t(x)\) because \(\deg s(x) < \deg t(x)\). Equation (4.58) can then be written as

$$x^K (s(x)M_U(x) + t(x)E_U(x)) = V(x).$$  \hspace{1cm} (4.59)

Observing the left hand side of (4.59), we know that all the terms on the right hand side of (4.59) of degree less than \(K\) will vanish. Thus, we have the following equivalent expression for (4.59):

$$s(x)M_U(x) + t(x)E_U(x) = V_U(x)$$  \hspace{1cm} (4.60)
where \( V_U(x) \triangleq \sum_{\ell=0} V_{K+\ell} x^\ell \) has degree
\[
\deg V_U(x) = \deg V(x) - K \\
\leq (K - 1) + \deg t(x) - K \\
< \deg t(x). \tag{4.61}
\]
Comparing (4.60) with (4.46) and from (4.61), we clearly have \( \deg r(x) = \deg V_U(x) < \deg t(x) \) which holds between lines 20 and 21 in the \( \mu \)-th loop iteration, and which coincides with (4.14). Thus, the correctness of (4.14) as a terminating condition is guaranteed (because from (4.56) \( \deg r(x) > (N - K)/2 \) holds between lines 13 and 14). On the other hand, since from (4.56) \( \deg r(x) > (N - K)/2 \) holds between lines 13 and 14 but \( \deg r(x) < \deg t(x) = \deg \Lambda_f(x) \leq (N - K)/2 \) holds between lines 20 and 21, we thus conclude that (4.15) can serve as an alternative terminating condition.
Chapter 5

An Algorithm for Simultaneous Partial Inverses

The simultaneous partial-inverse problem introduced in Chapter 1 is recalled below and investigated in detail. The major result here is the new algorithm for solving the problem. The proof of the algorithm also provides a different insight to the partial-inverse algorithm of Chapter 2. The applications to decoding Reed-Solomon codes and interleaved Reed-Solomon codes, beyond half the minimal distance, will be demonstrated in the following chapters.

5.1 The Simultaneous Partial-Inverse Problem

In this chapter, we propose a new algorithm that solves the problem.

Simultaneous Partial-Inverse (SPI) Problem: For \( i = 1, 2, \ldots, L \), let \( b^{(i)}(x) \) and \( m^{(i)}(x) \) be nonzero polynomials over some field \( F \) with \( \deg b^{(i)}(x) < \deg m^{(i)}(x) \). The problem is to find a nonzero polynomial \( \Lambda(x) \in F[x] \) of the smallest degree such that

\[
\deg \left( b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x) \right) < \tau^{(i)}
\]  

(5.1)
for given \( \tau^{(i)} \in \mathbb{Z} \) with \( 1 \leq \tau^{(i)} \leq \deg m^{(i)}(x) \).

It is not hard to prove (see Section 5.2) that this problem always has a unique solution, up to a scale factor.

In the special case where \( L = 1 \), the SPI problem reduces to the Partial-Inverse Problem of Chapter 2, which includes computing inverses in \( F[x]/m(x) \), computing Padé approximants, and solving the key equation for decoding Reed-Solomon codes as special cases.

The SPI problem for \( L > 1 \) is similar to the multi-sequence shift-register synthesis (MSSRS) problem \([37, 40]\) and can be used for similar purposes (such as \([41, 43]\)). Indeed, inspired by \([43, 45]\), we demonstrate the applications of the proposed SPI algorithm to decoding a scheme of interleaved Reed-Solomon codes beyond half the minimum distance. However, the SPI problem is not identical to the MSSRS problem; e.g., the SPI problem has always a unique solution but neither the original LFSRS problem \([2]\) nor the MSSRS problem has this property.

Also, the algorithm proposed in this paper looks very similar to the MSSRS algorithms in \([37, 40]\), but it is not identical to any of them. Moreover, the proof of the proposed algorithm is a nontrivial generalization of the proof in Chapter 2 and does not resemble any of the proofs of MSSRS algorithms.

The chapter is structured as follows. The existence, uniqueness, and degree of a solution of the SPI problem are proved in Section 5.2. The new algorithm is proposed in Section 5.3 and its complexity is briefly discussed in Section 5.4.1. Some remarks are given in Section 5.4.2. The correctness of the algorithm is proved in Section 5.5. Finally, in Section 5.6 we conclude the chapter.

The following notation will be used. The coefficient of \( x^d \) of a polynomial \( b(x) \in F[x] \) will be denoted \( b_d \). The leading coefficient (i.e., the coefficient of \( x^{\deg b(x)} \)) of a nonzero polynomial \( b(x) \) will be denoted by \( \text{lcf}(b(x)) \), and we also define \( \text{lcf}(0) \triangleq 0 \). We will use “mod” both as in \( r(x) = b(x) \) mod \( m(x) \) (the remainder of a division) and as in \( b(x) \equiv r(x) \) mod \( m(x) \) (a congruence modulo \( m(x) \)).

### 5.2 Existence, Uniqueness, and Degree of The Solution

The existence of a solution to the SPI problem is obvious from noting that \( \Lambda(x) \triangleq \text{lcm}(m^{(1)}(x), \ldots, m^{(L)}(x)) \), the least common multiple of all
5.2 Existence, Uniqueness, and Degree of The Solution

Let $m^{(i)}(x)$, satisfies

$$b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x) = 0; \quad (5.2)$$

consequently, the solution $\Lambda(x)$ satisfies

$$\deg \Lambda(x) \leq \deg \left( \operatorname{lcm} \left( m^{(1)}(x), \ldots, m^{(L)}(x) \right) \right). \quad (5.3)$$

The following bound on the degree of the solution $\Lambda(x)$ will be proved in Section 5.5.2. (The bound can also be proved directly as in [6].)

**Theorem 5.1** The solution $\Lambda(x)$ of the Simultaneous Partial-Inverse Problem satisfies

$$\deg \Lambda(x) \leq \sum_{i=1}^{L} \left( \deg m^{(i)}(x) - \tau^{(i)} \right). \quad (5.4)$$

As for uniqueness, we have

**Proposition 5.1 (Uniqueness of Solution)** The solution $\Lambda(x)$ of the Simultaneous Partial-Inverse Problem is unique up to a scale factor.

**Proof:** Let $\Lambda'(x)$ and $\Lambda''(x)$ be two solutions of the problem, which implies $\deg \Lambda'(x) = \deg \Lambda''(x) \geq 0$. Define

$$r^{(i)}(x) \overset{\triangle}{=} b^{(i)}(x)\Lambda'(x) \mod m^{(i)}(x) \quad (5.5)$$

$$r^{(i)}(x) \overset{\triangle}{=} b^{(i)}(x)\Lambda''(x) \mod m^{(i)}(x) \quad (5.6)$$

and consider

$$\Lambda(x) \overset{\triangle}{=} \left( \operatorname{lcf} \Lambda''(x) \right)\Lambda'(x) - \left( \operatorname{lcf} \Lambda'(x) \right)\Lambda''(x). \quad (5.7)$$

Then

$$r^{(i)}(x) \overset{\triangle}{=} b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x) \quad (5.8)$$

$$= \left( \operatorname{lcf} \Lambda''(x) \right) r^{(i)}(x) - \left( \operatorname{lcf} \Lambda'(x) \right) r^{(i)}(x) \quad (5.9)$$

by the natural ring homomorphism $F[x] \to F[x]/m^{(i)}(x)$. Clearly, (5.9) implies that $\Lambda(x)$ also satisfies (5.1) for every $1 \leq i \leq L$. But (5.7) implies $\deg \Lambda(x) < \deg \Lambda'(x)$, which is a contradiction unless $\Lambda(x) = 0$. Thus $\Lambda(x) = 0$, which means that $\Lambda'(x)$ and $\Lambda''(x)$ are equal up to a scale factor. \(\square\)
5.3 The Proposed Algorithm

The SPI problem as stated in Section 5.1 can be solved by the algorithm.

Simultaneous Partial-Inverse (SPI) Algorithm:

**Input:** $m^{(i)}(x), b^{(i)}(x), \tau^{(i)}$ for $i = 1, \ldots, L$.

**Output:** $\Lambda(x)$ as in the problem statement.

1. for $i = 1, \ldots, L$ begin
2. \hspace{1em} $\Lambda^{(i)}(x) := 0$
3. \hspace{1em} $d^{(i)} := \deg m^{(i)}(x)$
4. \hspace{1em} $\kappa^{(i)} := \text{lcf } m^{(i)}(x)$
5. end
6. $\Lambda(x) := 1$
7. $\delta := \max_{i \in \{1, \ldots, L\}} \left( \deg m^{(i)}(x) - \tau^{(i)} \right)$
8. $i := 1$
9. loop begin
10. \hspace{1em} repeat
11. \hspace{2em} if $i > 1$ begin $i := i - 1$ end
12. \hspace{1em} else begin
13. \hspace{2em} if $\delta \leq 0$ return $\Lambda(x)$
14. \hspace{2em} $i := L$
15. \hspace{2em} $\delta := \delta - 1$
16. end
17. \hspace{1em} $d := \delta + \tau^{(i)}$
18. \hspace{1em} $\kappa := \text{coefficient of } x^d \text{ in } b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x)$
19. \hspace{1em} until $\kappa \neq 0$
20. if $d < d^{(i)}$ begin
21. \hspace{1em} swap $(\Lambda(x), \Lambda^{(i)}(x))$
22. \hspace{1em} swap $(d, d^{(i)})$
23. \hspace{1em} swap $(\kappa, \kappa^{(i)})$
24. \hspace{1em} $\delta := d - \tau^{(i)}$
25. end
26. $\Lambda(x) := \kappa^{(i)}\Lambda(x) - \kappa x^{d - d^{(i)}} \Lambda^{(i)}(x)$
27. end

Lines 1–8 are for initialization. The nontrivial part begins with Line 9. Note that lines 21–23 simply swap $\Lambda(x)$ with $\Lambda^{(i)}(x)$, $d$ with
5.3 The Proposed Algorithm

\[ \delta_{\text{max}} \]

\[ \delta_{\text{max}} - 1 \]

\[ i \]

Figure 5.1: Illustration of (5.11) and (5.12) for \( i_{\text{max}} = 2 \).

The only actual computations are in lines 18 and 26. Note that in line 18 we have \( \kappa = 0 \) if \( d \geq \deg m^{(i)}(x) \).

We now explain how the algorithm works. For any nonzero \( \Lambda(x) \) and any \( i \in \{1, 2, \ldots, L\} \), define

\[
rd^{(i)}(\Lambda) \triangleq \deg(b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x)), \quad (5.10)
\]

\[
\delta_{\text{max}}(\Lambda) \triangleq \max_{i \in \{1, \ldots, L\}} \left( rd^{(i)}(\Lambda) - \tau^{(i)} \right), \quad (5.11)
\]

and

\[
i_{\text{max}}(\Lambda) \triangleq \max \argmax_{i \in \{1, \ldots, L\}} \left( rd^{(i)}(\Lambda) - \tau^{(i)} \right), \quad (5.12)
\]

the largest among the indices \( i \) that maximize \( rd^{(i)}(\Lambda) - \tau^{(i)} \), cf. Figure 1.

At any given time, the algorithm works on the polynomial \( \Lambda(x) \). The inner \textbf{repeat} loop (lines 10–19) computes the quantities defined in (5.10)–(5.12): between lines 19 and 20 we have

\[
i = i_{\text{max}}(\Lambda), \; \delta = \delta_{\text{max}}(\Lambda), \; d = rd^{(i)}(\Lambda), \quad (5.13)
\]

and also

\[
\kappa = \text{lcf} \left(b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x)\right). \quad (5.14)
\]

In particular, the very first execution of the \textbf{repeat} loop (with \( \Lambda(x) = 1 \)) yields

\[
i = \max \argmax_{i \in \{1, \ldots, L\}} \left( \deg b^{(i)} - \tau^{(i)} \right), \quad (5.15)
\]
\[ d = \deg b^{(i)}(x), \text{ and } \kappa = \lcf b^{(i)}(x) \] between lines 19 and 20.

In the special case \( L = 1 \), lines 11–17 (excluding line 13) amount to \( d := d - 1 \); the algorithm reduces to the Partial-Inverse algorithm of Chapter 2.

The only exit from the algorithm is line 13. Since \( \delta \geq \delta_{\max}(\Lambda) \), the condition \( \delta \leq 0 \) guarantees that \( \Lambda(x) \) satisfies (5.1).

The algorithm maintains the auxiliary polynomials \( \Lambda^{(i)}(x) \), for \( i = 1, \ldots, L \), which are all initialized to \( \Lambda^{(i)}(x) = 0 \). Thereafter, however, \( \Lambda^{(i)}(x) \) become nonzero (after their first respective executions of lines 21–23) and satisfy

\[ i_{\max}(\Lambda^{(i)}) = i. \] (5.16)

The heart of the algorithm is line 26, which cancels the leading term in

\[ b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x) \] (5.17)

(except when the sequence \( i \) is visited for the very first time, see below). Line 26 is explained by the following lemma.

**Lemma 5.1 (Core of the Algorithm)** Let \( \Lambda'(x) \) and \( \Lambda''(x) \) be nonzero polynomials such that \( i \triangleq i_{\max}(\Lambda') = i_{\max}(\Lambda'') \) and \( \rd^{(i)}(\Lambda') \geq \rd^{(i)}(\Lambda'') \). Then \( \delta_{\max}(\Lambda') \geq \delta_{\max}(\Lambda'') \) and the polynomial

\[ \Lambda(x) \triangleq \kappa'' \Lambda'(x) - \kappa' x^{d''} \Lambda''(x) \] (5.18)

with \( d' \triangleq \rd^{(i)}(\Lambda') \), \( \kappa' \triangleq \lcf(b^{(i)}(x)\Lambda'(x) \mod m^{(i)}(x)) \), \( d'' \triangleq \rd^{(i)}(\Lambda'') \), and \( \kappa'' \triangleq \lcf(b^{(i)}(x)\Lambda''(x) \mod m^{(i)}(x)) \) satisfies both

\[ \rd^{(i)}(\Lambda) < \rd^{(i)}(\Lambda') \] (5.19)

and

\[ \delta_{\max}(\Lambda) \leq \delta_{\max}(\Lambda') \] (5.20)

and either

\[ i_{\max}(\Lambda) < i_{\max}(\Lambda'), \] (5.21)

or

\[ \delta_{\max}(\Lambda) < \delta_{\max}(\Lambda'). \] (5.22)

The lemma is proved in Section 5.5.1. It follows from (5.19)–(5.22) that the algorithm makes progress and eventually terminates.
For each index \( i \in \{1, \ldots, L\} \), when line 26 is executed for the very first time, it is necessarily preceded by the swap in lines 21–23. In this case, line 26 reduces to

\[
\Lambda(x) := -\left(\text{lcf} \ m^{(i)}(x)\right) x^{\deg m^{(i)}(x) - \text{rd}^{(i)}(\Lambda')} \Lambda'(x)
\]

where \( \Lambda'(x) \) is the value of \( \Lambda(x) \) before the swap. It follows, in particular, that \( \deg \Lambda(x) > \deg \Lambda'(x) \).

In any case, we always have

\[
\deg(b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x)) < d,
\]

after executing line 26.

Finally, we note that every execution of the swap in lines 21–23 strictly reduces \( d^{(i)} \). We also note the execution of line 24 results in

\[
\delta = \begin{cases} 
\delta_{\text{max}}(\Lambda), & \text{if } \Lambda(x) \neq 0 \\
\deg m^{(i)} - \tau^{(i)}, & \text{if } \Lambda(x) = 0
\end{cases}
\]

where the second case happens only once—the very first time—for each index \( i \in \{1, \ldots, L\} \).

### 5.4 Remarks

#### 5.4.1 Complexity of The Algorithm

The complexity of every iteration of lines 9–27 is dominated by the complexity of the inner repeat loop. Let \( M(n) \) denote the complexity of the inner loop, where \( n \triangleq \max \deg m^{(i)}(x) \). Due to (5.19)–(5.22) (Lemma 5.1), the algorithm executes at most \( O(Ln) \) iterations of the outer loop. It follows that the overall complexity of the algorithm is \( O(LnM(n)) \).

As for the complexity of line 18, we first note that

\[
\deg(b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x)) \leq d
\]

is guaranteed before every execution of line 18.

In the special case where \( m^{(i)}(x) = x^{\deg m^{(i)}(x)} \) for all \( i \in \{1, \ldots, L\} \) (where \( \deg m^{(i)}(x) \) need not to be equal), line 18 amounts to

\[
\kappa := b^{(i)}_d \Lambda_0 + b^{(i)}_{d-1} \Lambda_1 + \ldots + b^{(i)}_{d-\nu} \Lambda_\nu
\]
where $\nu \triangleq \deg \Lambda(x)$ and where $b^{(i)}_{\mu} \triangleq 0$ for $\mu < 0$. In another special case where $m^{(i)}(x) = x^{\deg m^{(i)}(x)} - 1$ for all $i$, line 18 becomes

$$\kappa := b^{(i)}_d \Lambda_0 + b^{(i)}_{[d-1]} \Lambda_1 + \ldots + b^{(i)}_{[d-\nu]} \Lambda_\nu$$

with $b^{(i)}_{[\mu]} \triangleq b^{(i)}_{\mu \mod n}$. In both cases, the computation of line 18 only requires $O(n)$ operations; the algorithm then has the complexity $O(Ln^2)$, which agrees with the complexity of the MSSRS algorithms in [38,39].

### 5.4.2 By-products

The proposed algorithm (of Section 5.3) produces also some nice by-products. In particular, we have the following two theorems.

**Theorem 5.2 (Redundancy Detection)** When the algorithm terminates, if we still have $d^{(i)} = \deg m^{(i)}(x)$ for any $i$, then $b^{(i)}(x)$ and $m^{(i)}(x)$ can be removed from the SPI problem without affecting its solution.

The proof is given in Section 5.5.2. We also have following stronger version of Theorem 5.1

**Theorem 5.3** When the algorithm terminates, we have

$$\deg \Lambda(x) = \sum_{i=1}^{L} \left( \deg m^{(i)}(x) - d^{(i)} \right)$$

(5.27)

where $\tau^{(i)} \leq d^{(i)} \leq \deg m^{(i)}(x)$ for every $i \in \{1, \ldots, L\}$.

The proof is given in Section 5.5.2.

### 5.5 Proof of The Algorithm

#### 5.5.1 Proof of Lemma 5.1

First, $\delta_{\max}(\Lambda') \geq \delta_{\max}(\Lambda'')$ is obvious from the assumptions. From (5.18), we obtain

$$r(x) \triangleq b^{(i)}(x) \Lambda(x) \mod m^{(i)}(x)$$

$$= \kappa'' r^{(i)}(x) - \kappa' x^{d''} r''(x)$$

(5.29)
where
\[
\begin{align*}
  r'(x) & \triangleq \frac{b^{(i)}(x)\Lambda'(x)}{m^{(i)}(x)}, \\
normalized{r''(x)} & \triangleq \frac{b^{(i)}(x)\Lambda''(x)}{m^{(i)}(x)},
\end{align*}
\]
(5.30) (5.31)
by the natural ring homomorphism \( F[x] \to F[x]/m^{(i)}(x) \). It is then obvious from (5.29) that \( \deg r(x) < \deg r'(x) = d' \), which is (5.19).

For the remaining proof, we define
\[
\delta^{(\ell)}(\Lambda) \triangleq r^{(\ell)}(\Lambda) - \tau^{(\ell)}
\]
(5.32)
for every \( \ell \in \{1, \ldots, L\} \). With this notation, we have
\[
\delta^{(i)}(\Lambda) < \delta^{(i)}(\Lambda')
\]
(5.33)
from (5.19). We will next show that
\[
\delta^{(j)}(\Lambda) \leq \delta^{(i)}(\Lambda') \text{ for } j < i
\]
(5.34)
and
\[
\delta^{(k)}(\Lambda) < \delta^{(i)}(\Lambda') \text{ for } k > i.
\]
(5.35)
Clearly, (5.33)–(5.35) together imply both (5.20) and either (5.21) or (5.22) (or both).

To this end, we first note that \( d' - d'' = \delta^{(i)}(\Lambda') - \delta^{(i)}(\Lambda'') \), and thus
\[
d' - d'' + \delta^{(i)}(\Lambda'') = \delta^{(i)}(\Lambda').
\]
(5.36)
We then note from (5.18) that
\[
\delta^{(\ell)}(\Lambda) \leq \max \left\{ \delta^{(\ell)}(\Lambda'), d' - d'' + \delta^{(\ell)}(\Lambda'') \right\}
\]
(5.37)
for every \( \ell \in \{1, \ldots, L\} \).

Concerning (5.34), the assumption \( i_{\max}(\Lambda') = i \) implies that
\[
\delta^{(j)}(\Lambda') \leq \delta^{(i)}(\Lambda')
\]
(5.38)
for every \( j < i \). Moreover, \( i_{\max}(\Lambda'') = i \) implies that \( \delta^{(j)}(\Lambda'') \leq \delta^{(i)}(\Lambda'') \) and thus
\[
d' - d'' + \delta^{(j)}(\Lambda'') \leq d' - d'' + \delta^{(i)}(\Lambda'').
\]
(5.39)
It then follows from (5.37)–(5.39) that for all \( j < i \)
\[
\delta^{(j)}(\Lambda) \leq \max \left\{ \delta^{(i)}(\Lambda'), d' - d'' + \delta^{(i)}(\Lambda'') \right\}
\]
(5.40)
We then obtain (5.34) from (5.36).

Concerning (5.35), the assumption $i_{\text{max}}(\Lambda') = i$ implies

$$\delta^{(k)}(\Lambda') < \delta^{(i)}(\Lambda')$$

(5.41)

for every $k > i$. Moreover, $i_{\text{max}}(\Lambda'') = i$ implies that $\delta^{(k)}(\Lambda'') < \delta^{(i)}(\Lambda'')$ and thus

$$d' - d'' + \delta^{(k)}(\Lambda'') < d' - d'' + \delta^{(i)}(\Lambda'').$$

(5.42)

It then follows from (5.37), (5.41), and (5.42) that for all $k > i$

$$\delta^{(k)}(\Lambda) < \max \{\delta^{(i)}(\Lambda'), d' - d'' + \delta^{(i)}(\Lambda'')\}$$

(5.43)

We then obtain (5.35) from (5.36).

### 5.5.2 Annotated Algorithm

For the detailed proof, we annotate the algorithm of Section 5.3 with some extra variables and some assertions as follows.

1. for $i = 1, \ldots, L$ begin
2. \hspace{1em} $\Lambda^{(i)}(x) := 0$
3. \hspace{1em} $d^{(i)} := \deg m^{(i)}(x)$
4. \hspace{1em} $\kappa^{(i)} := \text{lcf } m^{(i)}(x)$
5. end
6. $\Lambda(x) := 1$
7. $\delta := \max_{i \in \{1, \ldots, L\}} \left( \deg m^{(i)}(x) - \tau^{(i)} \right)$
8. $i := 1$

Extra:

\begin{align*}
\hspace{1em} k &:= 0 \quad \text{(E.1)}
\end{align*}

9. loop begin

\begin{align*}
\text{Assertions:} \\
\deg \Lambda(x) &= \sum_{i=1}^{L} \left( \deg m^{(i)}(x) - d^{(i)} \right) \quad \text{(A.10)} \\
\deg \Lambda(x) &> \deg \Lambda^{(i)}(x), \quad i = 1, \ldots, L \quad \text{(A.11)}
\end{align*}

10. repeat

\hspace{1em} if $i > 1$ begin $i := i - 1$ end

\hspace{1em} else begin

\hspace{2em} if $\delta \leq 0$ return $\Lambda(x)$

\hspace{2em} $i := L$

\hspace{2em} $\delta := \delta - 1$

\hspace{1em} end

11. \end{loop}
Proof of The Algorithm

end

\[ d := \delta + \tau^{(i)} \]

\[ \kappa := \text{coefficient of } x^d \text{ in } b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x) \]

until \( \kappa \neq 0 \)

Assertion:
\[ i = i_{\text{max}}(\Lambda), \ \delta = \delta_{\text{max}}(\Lambda) \geq 0 \quad \text{(A.12)} \]

if \( d < d^{(i)} \) begin

Assertion:
\[ d^{(i)} > d \geq \tau^{(i)} \quad \text{(A.13)} \]

Extras:
\[ k := k + 1, \ i_k \triangleq i, \ \Lambda_k(x) \triangleq \Lambda(x), \]
\[ \Delta_k \triangleq d^{(i)} - d, \ d_k \triangleq d^{(i)} \quad \text{(E.2)} \]

swap \((\Lambda(x), \Lambda^{(i)}(x))\)

swap \((d, d^{(i)})\)

swap \((\kappa, \kappa^{(i)})\)

\[ \delta := d - \tau^{(i)} \]

Assertions:
\[ d > d^{(i)} \geq \tau^{(i)} \quad \text{(A.14)} \]
\[ \deg \Lambda^{(i)}(x) > \deg \Lambda(x) \quad \text{(A.15)} \]
\[ \deg \Lambda^{(i)}(x) > \deg \Lambda^{(j)}(x) \text{ for } j \neq i \quad \text{(A.16)} \]
\[ i_{\text{max}}(\Lambda^{(i)}) = i, \ \delta_{\text{max}}(\Lambda^{(i)}) \geq 0 \quad \text{(A.17)} \]

end

\[ \Lambda(x) := \kappa^{(i)}\Lambda(x) - \kappa x^d - d^{(i)}\Lambda^{(i)}(x) \]

Assertion:
\[ \text{rd}^{(i)}(\Lambda) < d \quad \text{(A.18)} \]
\[ \deg \Lambda(x) = \Delta_k + \deg \Lambda_k(x) \quad \text{(A.19)} \]
\[ > \deg \Lambda^{(i)}(x), \ i = 1, \ldots, L \quad \text{(A.20)} \]

end

We will prove these assertions iteration by iteration, and show that the algorithm terminates. Theorems \[\text{5.1} \] \[\text{5.3} \] are also proved below. The added extras \((E.1)\) and \((E.2)\) will help for proving the minimality of the returned \(\Lambda(x)\).

Assertions \((A.10)\) and \((A.11)\) are obvious from the initialization, and thus they are true in the first iteration. For the later iterations, they will be proved below by induction. We assume that they are true for the moment.
(A.12) is simply (5.13) as mentioned in Section 5.3, what the inner repeat loop aims to find; the part \( \delta \geq 0 \) is simply the contrary to the condition of line 13.

(A.13) is obvious, where \( d \geq \tau^{(i)} \) follows from \( \delta \geq 0 \) of (A.12).

Line 24 resets \( \delta := d - \tau^{(i)} \geq 0 \) for the next execution of the inner repeat loop.

(A.14)–(A.17) are from (A.11)–(A.13) followed by the swap in lines 21–23. Note that (A.14) implies the very strong fact.

**Proposition 5.2** Throughout the SPI algorithm, \( d^{(i)} \geq \tau^{(i)} \) holds for every \( i \in \{1, \ldots, L\} \).

As for (A.18), i.e., \( \deg(b^{(i)}(x)\Lambda(x) \mod m^{(i)}(x)) < d \), in the case where \( b^{(i)}(x) \) is visited for the very first time, we still have \( d = \deg m^{(i)}(x) \) after the very first execution of line 26, which makes (A.18) obvious. For the later executions of line 26 (A.18) follows from Lemma 5.1 (A.18) explains (5.24).

We now prove (A.19) and (A.20), which will lead to the subsequent (A.10) and (A.11). Note first that

**Lemma 5.2 (Degree Change Lemma)** Line 26 changes the degree of \( \Lambda(x) \) only in the iteration where lines 20–25 are executed.

The proof of Lemma 5.2 is given in Section 5.5.3. (A.19) then follows from that line 26 increases \( \deg \Lambda(x) \) as follows:

- Upon finishing an execution of lines 20–25, line 26 increases the degree of \( \Lambda(x) \) to

\[
\deg \Lambda^{(i)}(x) + d - d^{(i)} = \deg \Lambda_{k}(x) + \Delta_{k} \tag{5.44}
\]

since the \( \Lambda^{(i)}(x) \) is the one of (A.15) and (A.16), and is actually \( \Lambda_{k}(x) \) of (E.2), and \( d - d^{(i)} \) here is \( \Delta_{k} \) of (E.2). (A.20) is then obvious from (A.16).

- Subsequent iterations where the condition of line 20 does not hold (i.e., without involving an execution of lines 21–24) do not change the degree of \( \Lambda(x) \) by Lemma 5.2.

(A.20) leads to the subsequent (A.11), while (A.19) is actually equal to (A.10) of the subsequent iteration:
• \( \deg \Lambda^{(i)}(x) \) of (A.15) has the value

\[
\deg \Lambda^{(i)}(x) = \sum_{j \neq i}^{L} \left( \deg m^{(j)}(x) - d^{(j)} \right) + \deg m^{(i)}(x) - d
\]

(5.45)

(from (A.10) of the corresponding iteration).

• It is then clear that (5.44) is also equal to

\[
\deg \Lambda^{(i)}(x) + d - d^{(i)} = \sum_{j=1}^{L} \left( \deg m^{(j)}(x) - d^{(j)} \right),
\]

(5.46)

which gives (A.10) of the subsequent iteration.

By Lemma 5.2, the value of (A.10) is unchanged until the next execution lines 20–25.

The argument up to the point also shows that

**Proposition 5.3** \( \Lambda_k(x) \) of (E.2) satisfies

\[
\deg \Lambda_k(x) > \ldots > \deg \Lambda_2(x) > \deg \Lambda_1(x),
\]

(5.47)

with

\[
\deg \Lambda_{t+1}(x) = \Delta_t + \deg \Lambda_{t}(x)
\]

(5.48)

for \( 1 \leq t < k \), where \( \deg \Lambda_1(x) = 0 \) since \( \Lambda_1(x) = 1 \).

The proposition will play a key role in Section 5.5.4.

There are only finite iterations between any two executions of lines 20–25 because every execution of inner repeat loop (lines 10–19) strictly decreases \( \delta_{\max}(\Lambda) \) or \( i_{\max}(\Lambda) \) according to Lemma 5.1. The algorithm is then guaranteed to terminate since the swap in lines 21–23 strictly decreases \( d^{(i)} \).

Finally, we note that Theorems 5.1 and 5.3 follow from (A.10) and Proposition 5.2. As for Theorem 5.2, it follows from the observation: suppose that we have \( d^{(j)} = \deg m^{(j)}(x) \) for some \( j \) when the algorithm terminates. We know that when the algorithm terminates, the solution \( \Lambda(x) \) satisfies \( r_d^{(i)}(\Lambda) < \tau^{(i)} \) for every \( i \in \{1, \ldots, L\} \) including \( i = j \), but the condition \( d^{(j)} = \deg m^{(j)}(x) \) implies that \( \Lambda^{(j)}(x) = 0 \) has never been updated throughout the algorithm because \( i_{\max}(\Lambda) \) is never to be \( j \) until the algorithm stops; therefore, we can remove \( b^{(j)}(x) \) and \( m^{(j)}(x) \) from the SPI problem without affecting the solution \( \Lambda(x) \); indeed, \( d^{(j)} - \deg m^{(j)}(x) = 0 \) has no contribution to \( \deg \Lambda(x) \) of (A.10).
5.5.3 Proof of Lemma 5.2

We will prove the lemma by induction. Suppose that we have just finished the \(k\)-th execution of lines 20–25, and let \(i_{k,n}\) and \(\delta_{k,n}\) denote the respective values of \(i\) and \(d - d^{(t)}\) of the \(n\)-th execution of line 26 after the present and before the \((k + 1)\)-th execution of lines 20–25. We gather the executions of line 26 (between the \(k\)-th and \((k + 1)\)-th executions of lines 20–25) into a group, called the \(k\)-th group; clearly, for the \(k\)-th group, we have \(i_{k,1} = i_k\) and \(\delta_{k,1} = \Delta_k\) of (E.2).

To prove the lemma is equivalent to prove the statement: subsequent executions of line 26 in the present group do not change the degree of \(\Lambda(x)\).

As an inductive hypothesis, assume that \(\text{deg } \Lambda(x)\) only changes in the first execution of line 26 of every previous group, and assume that \(\text{deg } \Lambda(x)\) does not change in the present group for the first \(n - 1\) executions where \(n \geq 2\). We will verify the possible cases that might appear in the \(n\)-th execution of line 26 of the \(k\)-th group; these cases can be deduced from (5.19)–(5.22) of Lemma 1.

Note the fact that \(\Lambda^{(t)}(x)\) for any \(i\) is never changed inside the \(k\)-th group because where the algorithm updates only \(\Lambda(x)\).

Case 1: if \(i_{k,n} = i_{k,n-1}\), we have \(\delta_{k,n} < \delta_{k,n-1} \leq \delta_{k,1}\) because of (5.19). Together with the fact that \(\Lambda^{(i_{k,n})}(x)\) is never changed inside the \(k\)-th group, line 26 does not increase \(\text{deg } \Lambda(x)\); if \(L = 1\), the proof is completed here since this is the only case need to consider, as in Section 2.7.

In the following, we consider the possible different cases where \(i_{k,n} \neq i_{k,n-1}\). Note that if \(i_{k,n} \neq i_{k,n-1}\), the \(\Lambda^{(i_{k,n})}(x)\) involved in line 26 must be such that its first execution of line 26 has appeared in “some” previous group that is closest to the present \(k\)-th group; otherwise, this would contradict the premise of the equivalent statement that we want to prove.

Case 2: suppose \(i_{k,n} \neq i_{k,n-1}\), and let us say this is the very first revisit of the same \(i_{k,n}\) since that “some” previous group. We then note that in the present group, the net increase of \(d = rd^{(i_{k,n})}(\Lambda)\) is never larger than \(\delta_{k,1}\) because of (5.20). Together with the induction hypothesis that \(\text{deg } \Lambda(x)\) only changes in the first execution of every previous group, and thus that for every previous group, say \(\nu\)-th group, the net increase of \(rd^{(i_{k,n})}(\Lambda)\) in that group is never larger than the corresponding \(\delta_{\nu,1}\) in that group (because of (5.19)–(5.22)), we conclude that the accumulation of the net increase of \(rd^{(i_{k,n})}(\Lambda)\) from the “some”
to the present group is never larger than the corresponding accumulation of the net increase of deg \( \Lambda(x) \) from these groups. We therefore conclude that deg \( \Lambda(x) \) does not change in this case.

Case 3: suppose \( i_{k,n} \neq i_{k,n-1} \), and let us say that this is a second revisit of \( b^{(i_{k,n})}(x) \) in the present group. For this case, line 26 does not change deg \( \Lambda(x) \) because of (5.19)–(5.22) and of the same \( \Lambda^{(i_{k,n})}(x) \), which is never changed as in Case 1.

Case 4: suppose \( i_{k,n} \neq i_{k,n-1} \), and let us say that this is a first revisit of \( b^{(i_{k,n})}(x) \) in the present group, and say that \( b^{(i_{k,n})}(x) \) has been revisited in the groups between the “some” and the present one. This case can be viewed as continuing Cases 2 and 3, i.e., Cases 2 and 3 already appeared in the previous groups. But even though, we still have, in the present group, \( \delta_{k,n} \leq \delta_{k,1} \) from (5.19)–(5.22). Together with the inductive hypothesis that deg \( \Lambda(x) \) does not change in the present group for the first \( n-1 \) executions, and with the fact that \( \Lambda^{(i_{k,n})}(x) \) is not changed inside the \( k \)-th group, we learn that line 26 does not change deg \( \Lambda(x) \) in this case.

From Cases 2–4, we show that line 26 does not change deg \( \Lambda(x) \) if \( i_{k,n} \neq i_{k,n-1} \). Together with Case 1, we conclude that deg \( \Lambda(x) \) does not change in the \( n \)-th execution in the present group. The equivalent “statement” (and thus Lemma 5.2) then follows by induction.

5.5.4 Proving the Minimality of The Returned \( \Lambda(x) \)

To show that the returned \( \Lambda(x) \) is a polynomial of the smallest degree that satisfies (5.1) for all \( i \), it suffices to show that any nonzero \( \tilde{\Lambda}(x) \) with deg \( \tilde{\Lambda}(x) < \deg \Lambda(x) \) satisfies

\[
\deg(b^{(i)}(x)\tilde{\Lambda}(x) \mod m^{(i)}(x)) \geq \tau^{(i)}
\]

for some \( i \in \{1, \ldots, L\} \). The proof below essentially consists of three key steps.

We start with Proposition 5.3 and assume that \( \Lambda_k \) is the last one recorded in (E2). Note that the returned \( \Lambda(x) \) has deg \( \Lambda(x) > \deg \Lambda_k(x) \). We then have the lemma.

**Lemma 5.3 (The First Key: Decomposition)** Any nonzero \( \tilde{\Lambda}(x) \) with deg \( \tilde{\Lambda}(x) < \deg \Lambda(x) \) (where \( \Lambda(x) \) is the one returned by the algorithm) can be uniquely written as

\[
\tilde{\Lambda}(x) = \sum_{t=1}^{k} q_t(x)\Lambda_t(x)
\]
for some $q_t(x)$ with
\[
\deg q_t(x) < \deg \Lambda_{t+1}(x) - \deg \Lambda_t(x)
\] (5.51)
for $t = 1, \ldots, k - 1$, and
\[
\deg q_k(x) < \deg \Lambda(x) - \deg \Lambda_k(x).
\] (5.52)

The lemma is obvious by dividing $\tilde{\Lambda}(x)$ successively in order by
\[
\{\Lambda_k(x), \Lambda_{k-1}(x), \ldots, \Lambda_1(x) = 1\};
\]
$q_t(x)$ can be zero.

To show that any nonzero $\tilde{\Lambda}(x)$ with $\deg \tilde{\Lambda}(x) < \deg \Lambda(x)$ satisfies (5.49) for some $i$ is equivalent to show that any nonzero (5.50) satisfies $\delta_{\max}(\tilde{\Lambda}) \geq 0$. To this end, we need to study the respective values of $i_{\max}(q_t \Lambda_t)$ and $\delta_{\max}(q_t \Lambda_t)$, cf. Lemma 5.4 below; we have known from (A.12) and (E.2) that
\[
i_{\max}(\Lambda_t) = i_t, \quad \delta_{\max}(\Lambda_t) \geq 0
\] (5.53)
for all $t \leq k$.

We note from Proposition 5.3 that
\[
\Delta_t = \deg \Lambda_{t+1}(x) - \deg \Lambda_t(x)
\] (5.54)
for $t = 1, \ldots, k - 1$; moreover, we have (associated with the returned $\Lambda(x)$)
\[
\Delta_k = \deg \Lambda(x) - \deg \Lambda_k(x).
\] (5.55)

We then obtain from (5.51)–(5.55) that
\[
\deg q_t(x) < \Delta_t
\] (5.56)
for $t = 1, 2, \ldots, k$.

On the other hand, we have from (E.2) that
\[
\Delta_t = d_t - \deg \left( b^{(i_t)}(x)\Lambda_t(x) \mod m^{(i_t)}(x) \right)
\] (5.57)
for $t = 1, 2, \ldots, k$. We then obtain from (5.56) and (5.57) that
\[
\deg q_t(x) + \deg \left( b^{(i_t)}(x)\Lambda_t(x) \mod m^{(i_t)}(x) \right) < d_t
\] (5.58)
for $t = 1, 2, \ldots, k$. The following lemma then follows from (5.58) and the fact $d_t \leq \deg m^{(i_t)}(x)$. 
Lemma 5.4 (Leading Property of $q_t(x)\Lambda_t(x)$) Any nonzero term $q_t(x)\Lambda_t(x)$ of (5.50) satisfies

\[ i_{\text{max}}(q_t\Lambda_t) = i_{\text{max}}(\Lambda_t) \tag{5.59} \]

and

\[ \delta_{\text{max}}(q_t\Lambda_t) = \deg q_t(x) + \delta_{\text{max}}(\Lambda_t). \tag{5.60} \]

By now, we have done the first key step. (5.58)–(5.60) are the main results of this step.

We begin the second key step with the observation $i_{\text{max}}(\Lambda_t) = i_t = i$ for some $i \in \{1, \ldots, L\}$. We can therefore divide these $\Lambda_t(x)$ into (at most) $L$ different sets, say $S_1, \ldots, S_L$, such that

\[ S_i \triangleq \{ t : i_{\text{max}}(\Lambda_t) = i \}, \tag{5.61} \]

and then rewrite (5.50) as follows

\[ \tilde{\Lambda}(x) = \sum_{i=1}^{L} \left( \sum_{t \in S_i} q_t(x)\Lambda_t(x) \right) \tag{5.62} \]

\[ = \sum_{i=1}^{L} \tilde{\Lambda}^{(i)}(x) \tag{5.63} \]

where

\[ \tilde{\Lambda}^{(i)}(x) \triangleq \sum_{t \in S_i} q_t(x)\Lambda_t(x) \tag{5.64} \]

if $S_i$ is a nonempty set; otherwise, $\tilde{\Lambda}^{(i)}(x) \triangleq 0$ if $S_i$ empty.

If (5.63) is nonzero, some of $\tilde{\Lambda}^{(i)}(x)$ must be nonzero.

Lemma 5.5 (The Second Key: Leading Property of $\tilde{\Lambda}^{(i)}$) Any nonzero polynomial (5.64) satisfies

\[ i_{\text{max}}(\tilde{\Lambda}^{(i)}) = i \tag{5.65} \]

and

\[ \delta_{\text{max}}(\tilde{\Lambda}^{(i)}) \geq 0. \tag{5.66} \]
The proof is given below. We have done the second step. We are now ready to show that any nonzero (5.50) (i.e., (5.63)) will satisfy $\delta_{\text{max}}(\tilde{\Lambda}) \geq 0$. Note that the mapping

$$
\psi : F[x] \rightarrow F[x]/m^{(1)}(x) \times \ldots \times F[x]/m^{(L)}(x) : \quad a(x) \mapsto \psi(a) \triangleq (\psi_1(a), \ldots, \psi_L(a))
$$

with $\psi_i(a) \triangleq b^{(i)}(x)a(x) \mod m^{(i)}(x)$ satisfies

$$
\psi(\tilde{\Lambda}) = \psi \left( \sum_{i=1}^{L} \tilde{\Lambda}^{(i)}(x) \right) = \sum_{i=1}^{L} \psi(\tilde{\Lambda}^{(i)}).
$$

Note also that Figure 5.2 serves, in some sense, also a nice illustration of $\psi(\tilde{\Lambda}^{(i)})$ for $i = 4$. It is then clear that $\psi(\tilde{\Lambda})$ is completely characterized by the superposition of the figures of this sort. Specifically, the following lemma characterizes $\delta_{\text{max}}(\tilde{\Lambda})$ and $i_{\text{max}}(\tilde{\Lambda})$ of (5.63).

**Lemma 5.6 (The Last Key)** Let $S \subseteq \{1, \ldots, L\}$ be a nonempty set. For any $i \in S$, assume that $\tilde{\Lambda}^{(i)}(x)$ is a nonzero polynomial such that $i_{\text{max}}(\tilde{\Lambda}^{(i)}) = i$ and $\delta_{\text{max}}(\tilde{\Lambda}^{(i)}) \geq 0$. Then, for any $S$, the polynomial

$$
\tilde{\Lambda}(x) \triangleq \sum_{j \in S} \tilde{\Lambda}^{(j)}(x)
$$

satisfies

$$
\delta_{\text{max}}(\tilde{\Lambda}) = \max_{j \in S} \{\delta_{\text{max}}(\tilde{\Lambda}^{(j)})\} \geq 0,
$$

Figure 5.2: Illustration of (5.65) and (5.66) of Lemma 5.5 for $\tilde{\Lambda}^{(4)}(x)$. 

\begin{align*}
\delta_{\text{max}} & \quad \delta_{\text{max}} - 1 \\
1 & \quad 2 \quad 3 \quad i_{\text{max}} \quad 5 \quad i
\end{align*}
\[ i_{\text{max}}(\tilde{\Lambda}) = \max \arg\max_{j \in S} \{ \delta_{\text{max}}(\tilde{\Lambda}(j)) \}, \quad (5.71) \]

and also \( \tilde{\Lambda}(x) \neq 0 \).

**Proof:** The proof of (5.70) and (5.71) follows from that (5.67) satisfies

\[ \psi(\tilde{\Lambda}) = \sum_{j \in S} \psi(\tilde{\Lambda}(j)) \quad (5.72) \]

for any nonempty \( S \), and that every element \( \tilde{\Lambda}(j)(x) \) involved in this set \( \max_{j \in S} \{ \delta_{\text{max}}(\tilde{\Lambda}(j)) \} \) has different \( i_{\text{max}} \). Consequently, we have \( \delta_{\text{max}}(\tilde{\Lambda}) \geq 0 \) and thus \( \tilde{\Lambda}(x) \neq 0 \). \( \square \)

It then follows from Lemma 5.6 that any nonzero (5.63) (namely, (5.50)) satisfies \( \delta_{\text{max}}(\tilde{\Lambda}) \geq 0 \), and thus satisfies (5.49) for some \( 1 \leq i \leq L \), which concludes the proof of the minimality of the returned \( \Lambda(x) \).

**Remark:** We do not need Lemma 5.6 if \( L = 1 \); for such case, the proof also serves as an alternative proof to that of Section 2.7.

**Proof of Lemma 5.5:** Recall that \( S_i \) of nonzero (5.64) is a nonempty set such that \( i_{\text{max}}(\Lambda_t) = i \) for any \( t \in S_i \), and recall (E.2). We then note that for any \( t, t' \in S_i \) with \( t < t' \), we have

\[ \deg m^{(i)}(x) \geq d_t > d_{t'} \geq \tau^{(i)}, \quad (5.73) \]

which is self-evident from Section 5.5.2. It then follows from (5.58) that for any nonzero \( q_t(x)\Lambda_t(x) \), we have

\[ \deg q_t(x) + \deg \left( b^{(i)}(x)\Lambda_t(x) \mod m^{(i)}(x) \right) < d_t \quad (5.74) \]

and (with \( \deg q_t(x) = 0 \))

\[ \deg \left( b^{(i)}(x)\Lambda_t(x) \mod m^{(i)}(x) \right) \geq d_{t'}. \quad (5.75) \]

The argument then implies that the special \( q_{\tilde{t}}(x)\Lambda_{\tilde{t}}(x) \) with

\[ \tilde{t} \triangleq \min\{ t : q_t(x) \neq 0, t \in S_i \} \quad (5.76) \]

is the one, among all the nonzero \( q_t(x)\Lambda_t(x) \) with \( t \in S_i \), that yields the largest

\[ \text{rd}^{(i)}(q_t\Lambda_t) = \deg q_t(x) + \text{rd}^{(i)}(\Lambda_t) \geq \tau^{(i)} \quad (5.77) \]
Therefore, we have $\delta_{\max}(\tilde{\Lambda}^{(i)}) = \delta_{\max}(q_i\Lambda_i) \geq 0$.

Finally, we note that

$$i_{\max}(\tilde{\Lambda}^{(i)}) = i_{\max}(q_i\Lambda_i) \geq 0.$$

$$i_{\max}(\tilde{\Lambda}^{(i)}) = i_{\max}(q_i\Lambda_i) = i_{\max}(\Lambda_i) = i,$$

where the second step follows from (5.59).

5.6 Conclusion

We introduced the simultaneous partial-inverse (SPI) problem, which is similar, but not identical, to the multi-sequence shift-register synthesis (MSSRS) problem. We proposed a new algorithm to solve this problem, which is similar, but not identical, to known algorithms for MSSRS. The proof of the proposed algorithm does not resemble existing proofs of MSSRS algorithms. The applications of the SPI algorithm to decoding Reed-Solomon codes will be discussed in the later chapters.
Chapter 6

Simultaneous Partial-Inverse and Decoding Interleaved Reed-Solomon Codes

The partial-inverse approach that was introduced in Chapter 2 is further developed to decoding interleaved Reed-Solomon codes beyond half the minimum distance. The resulting algorithm is new. In particular, it is very efficient and practical even though the depth of interleaving is large.

6.1 Interleaved Reed-Solomon Codes

Let $F$ be a finite field, let $\beta_0, \ldots, \beta_{n-1}$ be $n$ different elements of $F$, let $m(x) \triangleq \prod_{\ell=0}^{n-1}(x - \beta_\ell)$, let $F[x]/m(x)$ be the ring of polynomials modulo $m(x)$, and let $\psi$ be the evaluation mapping

$$
\psi : F[x]/m(x) \rightarrow F^n : a(x) \mapsto (a(\beta_0), \ldots, a(\beta_{n-1})), \quad (6.1)
$$

which is a ring isomorphism. Note that $\deg m(x) = n$.

We then define an $[n, k]$ Reed-Solomon code $C$ with blocklength $n$ and dimension $k$ as the set

$$
C \triangleq \{c \in F^n : \deg \psi^{-1}(c) < k\}, \quad (6.2)
$$
Simultaneous Partial-Inverse and Decoding Interleaved Reed-Solomon Codes

and consider an interleaved Reed-Solomon (IRS) code $C_{IRS}$ of depth $L$ as a set such that every element of $C_{IRS}$ is a $L \times n$ matrix where every row corresponds to a codeword taken from $C$.

Such codes can be decoded by Feng-Tzeng algorithm or the MSSRS algorithms [37–40,43]. In the following, we develop a new algorithm for decoding such codes from the partial-inverse perspective.

6.2 Channel Model and Error Locator Polynomial

Now, let $A \in C_{IRS}$ denote a $L \times n$ matrix transmitted through a channel, and let

$$Y = A + E$$  \hspace{1cm} (6.3)

be the received matrix, where the matrix $E \in F^{L \times n}$ represents the error that corrupts $A$. Moreover, let $a(i)$ denote the $i$-th row of $A$, $y(i)$ the $i$-th row of $Y$, and $e(i)$ the $i$-th row of $E$. We then have

$$y(i) = a(i) + e(i),$$  \hspace{1cm} (6.4)

for $i = 1, \ldots, L$ and therefore

$$Y'(i)(x) = C'(i)(x) + E'(i)(x)$$  \hspace{1cm} (6.5)

where $Y'(i)(x) \triangleq \psi^{-1}(y(i))$, $C'(i)(x) \triangleq \psi^{-1}(a(i))$, and $E'(i)(x) \triangleq \psi^{-1}(e(i))$. Note that $\deg C'(i)(x) < k$ and $\deg E'(i)(x) < \deg m(x) = n$.

Clearly, every $C'(i)(x)$ can be decoded, independently, from the respective $Y'(i)(x)$ by using the Berlekamp-Massey or gcd-based decoding algorithms or by the PIA algorithm of Section 2.3 provided that the number of erroneous symbols in the $i$-th row is not larger than $(n - k)/2$.

However, if the errors tend to be concentrated in columns of $E$, we may prefer to consider column errors rather than symbol errors. Indeed, it has been shown in [42–45] that more than $(n - k)/2$ column errors can be corrected if $L > 1$.

Let $U \subset \{1, \ldots, n\}$ be the set indexing the nonzero columns of $E$. We then define, for any $E$, the error-locator polynomial

$$\Lambda_e(x) \triangleq \prod_{j \in U} (x - \beta_j).$$  \hspace{1cm} (6.6)

It is easy to see that $\Lambda_e(x)$ satisfies

$$E'(i)(x)\Lambda_e(x) \mod m(x) = 0$$  \hspace{1cm} (6.7)
for every \( i \in \{1, \ldots, L\} \). In consequence, we have
\[
\deg(Y^{(i)}(x)\Lambda_e(x) \mod m(x)) < k + \deg \Lambda_e(x)
\] (6.8)
for every \( i \).

### 6.3 Guaranteed Decoding Using Rank Information

It has been shown in [43–45] that the rank of the matrix \( E \) can be relevant for decoding IRS. In particular, every error pattern \( E \) with less than \( n - k \) column errors can be corrected provided that the rank of \( E \) equals \( |U| \).

Let \( r \) denote the rank of the submatrix formed by the nonzero columns of \( E \). We then have the following Theorem 6.1, which is a generalization of Theorem 2.3 of Section 2.5, and on the other hand is similar, but not identical, to Lemma 3 of [43]. Note that \( r \leq |U| = \deg \Lambda_e(x) \).

**Theorem 6.1 (A New Key Equation)** If
\[
2|U| \leq n - k + r - 1,
\] (6.9)
then the error locator polynomial (6.6) satisfies
\[
\deg(Y^{(i)}(x)\Lambda_e(x) \mod m(x)) < \frac{n + k + r - 1}{2}
\] (6.10)
for all \( i \in \{1, \ldots, L\} \). Conversely, for any \( Y \) and any \( E \in F^{L \times n} \) (of rank \( r \)) and \( t \in \mathbb{R} \) with
\[
|U| \leq t \leq \frac{n - k + r - 1}{2}
\] (6.11)
if some nonzero \( \Lambda(x) \in F[x] \) with \( \deg \Lambda(x) \leq t \) satisfies
\[
\deg(Y^{(i)}(x)\Lambda(x) \mod m(x)) < n - t + r - 1
\] (6.12)
for all \( i \in \{1, \ldots, L\} \), then \( \Lambda(x) \) is a multiple of \( \Lambda_e(x) \).

The proof is given in Section 6.6. In the special case where \( L = 1 \), we have \( r = 1 \) (if \( |U| > 0 \)) and the theorem reduces to Theorem 2.3 of Section 2.5.

In another special case where \( r = |U| \), (6.9) reduces to \( |U| < n - k \); in this case, \( t = r \) satisfies (6.11), and (6.12) becomes
\[
\deg(Y^{(i)}(x)\Lambda(x) \mod m(x)) < n - 1.
\] (6.13)
6.4 The New Decoding Algorithm

6.4.1 Locating Algorithm

Condition (6.8) and Theorem 6.1 suggest the following algorithm.

Error-Locating Algorithm:

1. Run the SPI algorithm of Section 5.3 with \( b^{(i)}(x) = Y^{(i)}(x) \), and \( m^{(i)}(x) = m(x) \), and \( \tau^{(i)} = n - 1 \) for every \( i \in \{1, \ldots, L\} \).

2. If the returned polynomial \( \Lambda(x) \) satisfies the condition
   \[
   \deg(Y^{(i)}(x)\Lambda(x) \mod m(x)) < k + \deg \Lambda(x)
   \]  
   for every \( i \in \{1, \ldots, L\} \), then stop.

3. Otherwise, decrease all \( \tau^{(i)} \) by 1 and continue the SPI algorithm.

4. Go to 2).

Theorem 6.1 guarantees that this algorithm finds \( \Lambda(x) = \gamma \Lambda_e(x) \) (for some nonzero \( \gamma \in F \)) provided that (6.9) is satisfied. Note that rank \( r \) of \( E \) is not needed in the error-locating algorithm. Also, the test (6.14) actually requires no extra computations; this error-locating method can be implemented by modifying the SPI algorithm of Section 5.3 as follows

SPI Error-Locating Algorithm:

The algorithm is the same as the SPI algorithm of Section 5.3 (with \( b^{(i)}(x) = Y^{(i)}(x) \), \( m^{(i)} = m(x) \), and \( \tau^{(i)} = n - 1 \), except that line 13 is replaced by following lines:

61 \begin{align*}
   &\text{if } \delta \leq 0 \text{ begin} \\
   &\quad \text{if } d \leq \deg \Lambda(x) + k \text{ return } \Lambda(x) \\
   &\quad \text{else begin} \\
   &\quad\quad \delta := \delta + 1 \\
   &\quad\quad \text{for } j = 1, \ldots, L \text{ begin } \tau^{(j)} := \tau^{(j)} - 1 \text{ end} \\
   &\quad\end{align*}

65 \end{align*}

Note that line 62 suffices to check (6.14) for all \( i \). Note also that \( \tau^{(1)} = \ldots = \tau^{(L)} \) throughout the algorithm.
In the special case where \( r = |U| \), it follows from (6.13) that the algorithm stops at the earliest possible moment (when (6.14) is checked for the first time); this special case is very likely if \( L \geq n - k \).

The error-locating algorithm is guaranteed to return \( \Lambda(x) = \gamma \Lambda_e(x) \) (for some nonzero \( \gamma \in F \)) if (6.9) is satisfied. This guarantee agrees with the guarantee in [43].

### 6.4.2 Decoding

Putting things together, we have the following decoding algorithm.

1. Compute \( Y^{(i)}(x) = \psi^{-1}(y^{(i)}) \) for all \( i \).

2. Run the above error-locating algorithm to obtain a candidate \( \Lambda(x) \) for the error locator polynomial.

3. Complete decoding in any standard way [10], or by means of

\[
C^{(i)}(x) = \frac{Y^{(i)}(x)\Lambda(x) \mod m(x)}{\Lambda(x)} \quad (6.15)
\]

as in Section 2.5 or by

\[
C^{(i)}(x) = Y^{(i)}(x) \mod \tilde{m}(x) \quad (6.16)
\]

where \( \tilde{m}(x) \) \( \triangleq m(x)/\Lambda(x) \), cf. Proposition 6.1 below.

If the division in (6.15) does not work out, or if \( \Lambda(x) \) does not divide \( m(x) \), or if the resulting polynomials \( C^{(i)}(x) \) do not satisfy \( \deg C^{(i)}(x) < k \), then a decoding failure should be declared.

The decoding algorithm can correct any \( E \) if (6.9) is satisfied. Note that any error \( E \) with \( r = |U| < n - k \) can be corrected.

The following proposition (nearly the same as Proposition 2.9) explains (6.16), which also applies to erasures-only decoding.

**Proposition 6.1** If \( \Lambda(x) = \gamma \Lambda_e(x) \) (for some nonzero \( \gamma \in F \)) satisfies \( \deg \Lambda(x) \leq n - k \), then

\[
C^{(i)}(x) = Y^{(i)}(x) \mod \tilde{m}(x) \quad (6.17)
\]

where \( \tilde{m}(x) \) \( \triangleq m(x)/\Lambda(x) \).
**Proof:** Note that $\tilde{m}(x)$ has degree $\deg \tilde{m}(x) \geq k > \deg C(i)(x)$. Note also that $\tilde{m}(x)$ divides $\gcd(E(i)(x), m(x))$ and thus $\tilde{m}(x)$ divides $E(i)(x)$. We then have

\[
Y(i)(x) \mod \tilde{m}(x) = C(i)(x) + E(i)(x) \mod \tilde{m}(x)
\]

(6.18)

\[
= C(i)(x).
\]

(6.19)

\[\square\]

### 6.5 A Remark on The Decoding Algorithm

In fact, the decoding algorithm (of Section 6.4.2) has the potential to correct errors up to

\[
|U| \leq \frac{L}{L+1}(n-k)
\]

(6.20)
due to the following observations.

First, we observe that if $\Lambda(x) = \Lambda_e(x)$ is a nonzero polynomial of the smallest degree that satisfies simultaneously for all $i$

\[
\deg \big(Y(i)(x)\Lambda(x) \mod m(x)\big) < k + |U|,
\]

(6.21)

then the error-locating algorithm (of Section 6.4.1) returns $\gamma\Lambda_e(x)$ for some nonzero $\gamma \in F$. In other words, the error-locating algorithm finds $\Lambda_e(x) = \gamma\Lambda_e(x)$ (for some nonzero $\gamma \in F$), if $\Lambda_e(x)$ is a solution of the SPI problem of $\tau(i) = k + |U|$ for all $i$.

Now, let’s assume that $\Lambda_e(x)$ of a given $E$ is the solution of the SPI problem of $\tau(i) = k + |U|$ for all $i$. Since such $\Lambda_e(x)$ with $\deg \Lambda_e(x) = |U|$ must satisfy $\deg \Lambda_e(x) \leq \sum_{i=1}^{L}(n-\tau(i))$ for $\tau(i) = k + |U|$ by Theorem 5.1 we then obtain (6.20).

The potential that the decoding algorithm may correct errors up to (6.20) then follows from the situation that $\Lambda_e(x)$ of given $E$ that satisfies (6.20) can possibly be the solution of the SPI problem of $\tau(i) = k + |U|$. The potential can be quantified by the probability of occurrence of such cases for given a statistical model of $E$.

### 6.6 Proof of Theorem 6.1

In the following, we prove Theorem 6.1. Note first that (6.9) implies

\[
k + |U| \leq n - |U| + r - 1
\]

\[
< n
\]

(6.22)
where the last step follows from $r \leq |U|$. We then obtain
\[
Y^{(i)}(x)\Lambda_e(x) \mod m(x) = C^{(i)}(x)\Lambda_e(x) \mod m(x) + E^{(i)}(x)\Lambda_e(x) \mod m(x) = C^{(i)}(x)\Lambda_e(x),
\]
where the last step follows from (6.7) and (6.22).

From (6.23), we have
\[
\deg\left(Y^{(i)}(x)\Lambda_e(x) \mod m(x)\right) < k + |U|,
\]
and (6.10) follows from $k + |U| \leq k + \frac{n-k+r-1}{2} = \frac{n+k+r-1}{2}$.

As for the converse, assume (6.11), (6.12), and $\deg \Lambda(x) \leq t$, and consider
\[
Y^{(i)}(x)\Lambda(x) \mod m(x) = C^{(i)}(x)\Lambda(x) + E^{(i)}(x)\Lambda(x) \mod m(x).
\]
Under the stated assumptions, the degree of the left-hand side of (6.25)
is smaller than $n - t + r - 1$ and also
\[
\deg\left(C^{(i)}(x)\Lambda(x)\right) < k + t \leq n - t + r - 1.
\]
It follows that
\[
\deg\left(E^{(i)}(x)\Lambda(x) \mod m(x)\right) < n - t + r - 1
\]
for all $i \in \{1, \ldots, L\}$.

Now, let $\tilde{Y}(x) \triangleq \sum_{i=1}^{L} Q^{(i)}Y^{(i)}(x)$ where $Q^{(i)} \in F$. We then have
\[
\tilde{Y}(x) = \tilde{C}(x) + \tilde{E}(x)
\]
where $\tilde{C}(x) \triangleq \sum_{i=1}^{L} Q^{(i)}C^{(i)}(x)$ satisfies $\deg \tilde{C}(x) < k$, and where
\[
\tilde{E}(x) \triangleq \sum_{i=1}^{L} Q^{(i)}E^{(i)}(x)
\]
satisfies
\[
\deg\left(\tilde{E}(x)\Lambda(x) \mod m(x)\right) < n - t + r - 1.
\]
Also, let $\tilde{e} \triangleq \psi(\tilde{E}(x)) = (\tilde{e}_1, \ldots, \tilde{e}_n) \in F^n$ with $\psi$ as in (6.1), and let $\tilde{U} \subset \{1, \ldots, n\}$ be the set indexing the nonzero entries of $\tilde{e}$. We
then have \( \tilde{e} = \sum_{i=1}^{L} Q^{(i)} e^{(i)} \) and \( \tilde{U} \subset U \) for any choice of \( Q^{(1)}, \ldots, Q^{(L)} \). Furthermore, we define

\[
\tilde{\Lambda}_e(x) \triangleq \prod_{j \in \tilde{U}} (x - \beta_j). \tag{6.31}
\]

Clearly, \( \deg \tilde{\Lambda}_e(x) = |\tilde{U}| \), and \( \tilde{\Lambda}_e(x) \) divides \( \Lambda_e(x) \).

Now we are ready to prove the converse part. Since \( E \) has rank \( r \), we can choose the values of \( Q^{(1)}, \ldots, Q^{(L)} \) such that \( \tilde{e} = \psi(\tilde{E}(x)) \in F^n \) has exact \( |\tilde{U}| = |U| - r + 1 \) nonzero entries for some \( \tilde{U} \subset U \). We then write

\[
\tilde{E}(x)\Lambda(x) = g(x)m(x) + \tilde{E}(x)\Lambda(x) \mod m(x) \tag{6.32}
\]

according to the polynomial division theorem. But \( \tilde{E}(x) \) (and thus \( \tilde{E}(x)\Lambda(x) \)) has at least

\[
n - |\tilde{U}| = n - |U| + r - 1 \geq n - t + r - 1 \tag{6.33}
\]

zeros in the set \{\( \beta_0, \beta_1, \ldots, \beta_{n-1} \)\}. It follows that \( \tilde{E}(x)\Lambda(x) \mod m(x) \) has also at least \( n - t + r - 1 \) zeros (in this set), which contradicts (6.30) unless

\[
\tilde{E}(x)\Lambda(x) \mod m(x) = 0. \tag{6.34}
\]

Therefore, \( \Lambda(x) \) satisfies (6.34).

But any such nonzero \( \Lambda(x) \) that satisfies (6.34) is a multiple of the polynomial \( \tilde{\Lambda}_e(x) \). It follows that \( \Lambda(x) \) must be a multiple of \( \tilde{\Lambda}_e(x) \) because \( \Lambda(x) \) is a multiple of \( \tilde{\Lambda}_e(x) \) for every \( \tilde{U} \subset U \) with \( |\tilde{U}| = |U| - r + 1 \).

### 6.7 Conclusion

We proposed a new key equation together with a new interpolation to decoding interleaved Reed-Solomon codes. The new key equation turns the decoding of such codes into a simultaneous partial-inverse problem, and enables the resulting algorithm to correct column errors beyond half the minimum distance. The resulting algorithm is new, and it is very efficient even if the depth of interleaving is large.
Chapter 7

Simultaneous Partial-Inverse and Reed-Solomon Decoding

In this chapter, we demonstrate the application of the proposed SPI algorithm to decoding a scheme of Reed-Solomon codes beyond half the minimal distance.

7.1 Decoding Beyond Half the Minimal Distance

Let $F$ be a finite field, let $\beta_0, \ldots, \beta_{n-1}$ be $n$ different elements of $F$, let $m(x) \triangleq \prod_{\ell=0}^{n-1} (x - \beta_\ell)$, and then let $C$ be an $[n, k]$ Reed-Solomon code defined, as in Section 6.1, as the set

$$
\{ c = (c_0, \ldots, c_n) \in F^n : \deg \psi^{-1}(c) < k \} \quad (7.1)
$$

with $\psi$ as in (6.1).

Let $y = (y_0, \ldots, y_{n-1}) \in F^n$ be the received word, which we wish to decompose into

$$
y = c + e \quad (7.2)
$$

where $c \in C$ is a codeword and where the Hamming weight of $e = (e_0, \ldots, e_{n-1}) \in F^n$ is as small as possible. Moreover, let $Y(x) \triangleq \psi^{-1}(y) = C(x) + E(x)$ where $C(x) \triangleq \psi^{-1}(c)$ and $E(x) \triangleq \psi^{-1}(e)$. 
For any \( e \in F^n \), we define the error locator polynomial
\[
\Lambda_e(x) \triangleq \prod_{\substack{\ell \in \{0, \ldots, n-1\} \\
e \ell \ne 0}} (x - \beta_\ell).
\] (7.3)

Clearly, \( \deg \Lambda_e(x) = w_H(e) \).

**Theorem 7.1** If \( w_H(e) \leq \frac{n-k}{2} \), then \( \Lambda(x) = \gamma \Lambda_e(x) \) is the unique polynomial, up to a nonzero scale factor \( \gamma \in F \), of the smallest degree that satisfies
\[
\deg \left( Y(x)\Lambda(x) \mod m(x) \right) < \frac{n+k}{2} \tag{7.4}
\]

The theorem is from Corollary 2.2, a consequence of the fact: if \( w_H(e) \leq \frac{n-k}{2} \) then any \( \Lambda(x) \neq 0 \) with \( \deg \Lambda(x) \leq (n-k)/2 \) that satisfies (7.4) is a nonzero multiple of \( \Lambda_e(x) \), cf. Theorem 2.3. The SPI algorithm with \( L = 1 \), \( b^{(1)}(x) = Y(x) \), \( m^{(1)}(x) = m(x) \), and \( \tau^{(1)} = [(n+k)/2] \), as in Chapter 2, will find such \( \gamma \Lambda_e(x) \), provided that \( w_H(e) \leq (n-k)/2 \).

In [41], Schmidt, Sidorenko, and Bossert proposed a scheme of decoding (low rate) Reed-Solomon codes beyond half the minimum distance, where finding the \( \Lambda_e(x) \) (possibly with \( \deg \Lambda_e(x) > (n-k)/2 \)) is formulated as solving a MSSRS problem (see also [40]). Specifically, the idea therein is to compute \( \Lambda_e(x) \) from a set of equations formed from the received \( Y(x) \) and a number of the power of \( Y(x) \). In the following, we show that this can also be done via solving a SPI problem of Section 5.1.

To this end, for \( i = 1, \ldots, L \), let
\[
C^{(i)}(x) \triangleq C^i(x) \mod m(x), \tag{7.5}
\]
\[
Y^{(i)}(x) \triangleq Y^i(x) \mod m(x), \tag{7.6}
\]
and let \( E^{(i)}(x) \triangleq Y^{(i)}(x) - C^{(i)}(x) \). We then note that
\[
E^{(i)}(x) = \left( (C(x) + E(x))^i - C^i(x) \right) \mod m(x)
= E(x)Q^{(i)}(x) \mod m(x) \tag{7.7}
\]
for some \( Q^{(i)}(x) \).

**Lemma 7.1** The polynomial \( \Lambda(x) = \gamma \Lambda_e(x) \) for any nonzero \( \gamma \in F \) satisfies
\[
Y^{(i)}(x)\Lambda(x) \mod m(x) = C^i(x)\Lambda(x) \tag{7.8}
\]
provided that \( i(k-1) + \deg \Lambda_e(x) < n \).
**Proof:** With $Y^{(i)}(x) = C^{(i)}(x) + E^{(i)}(x)$, the lemma is obvious from noting that

$$E^{(i)}(x)\Lambda(x) \mod m(x) = 0$$

(7.9)

because $E(x)\Lambda_e(x) \mod m(x) = 0$, and noting that

$$C^{(i)}(x)\Lambda(x) \mod m(x) = C^{(i)}(x)\Lambda(x)$$

(7.10)

because $\deg(C^{(i)}(x)\Lambda_e(x)) \leq i(k - 1) + \deg \Lambda_e(x) < n$. □

Now suppose that $\deg \Lambda_e(x) \leq t$, where $t \geq (n - k)/2$. (The exact value of $t$ where $(n - k)/2 \leq t \leq n - k$, and the corresponding value of $L$ will both be specified in the subsequent section.) Then, to find such $\gamma\Lambda_e(x)$ that simultaneously satisfies (7.8) for $i = 1, \ldots, L$ can be naturally treated as solving a SPI Problem with $b^{(i)}(x) = Y^{(i)}(x)$, $m^{(i)}(x) = m(x)$, and $\tau^{(i)} = i(k - 1) + t + 1$.

Note, however, that Lemma 7.1 does not have a converse; namely, here we do not have a theorem like Theorem 6.1 so there is no guarantee to find $\Lambda_e(x)$. Nevertheless, we do know from Lemma 7.1 the fact that the nonzero $\Lambda(x) = \gamma\Lambda_e(x)$ must satisfy

$$\deg(Y^{(i)}(x)\Lambda(x) \mod m(x)) < i(k - 1) + \deg \Lambda(x).$$

(7.11)

We can therefore use (7.11) as a test just like the test (6.14) in the error-locating algorithm of Section 6.4.1.

Putting things together, we obtain the error-locating algorithm.

**Error-Locating Algorithm:**

1. Compute $Y^{(i)}(x)$ for $i = 1, \ldots, L$.

2. Run the SPI algorithm of Section 5.3 with $b^{(i)}(x) = Y^{(i)}(x)$, and $m^{(i)}(x) = m(x)$, and $\tau^{(i)} = i(k - 1) + t + 1$ for every $i \in \{1, \ldots, L\}$.

3. If the returned polynomial $\Lambda(x)$ satisfies the condition (7.11) for every $i \in \{1, \ldots, L\}$, then stop.

4. Otherwise, decrease all $\tau^{(i)}$ by 1 and continue the SPI algorithm.

5. Go to 3).

□

Note again that $t$ and $L$ of the error-locating algorithm will be specified in the subsequent section. Also, the test (7.11) actually requires no extra computations; this error-locating method can be implemented by modifying the SPI algorithm of Section 5.3 as follows.
SPI Error-Locating Algorithm:

The algorithm is the same as the SPI algorithm of Section 5.3 (with \( b^{(i)}(x) = Y^{(i)}(x) \), \( m^{(i)} = m(x) \), and \( \tau^{(i)} = i(k - 1) + t + 1 \)), except that line 13 is replaced by following lines:

\[
\text{if } \delta \leq 0 \text{ begin}
\]

\[
\text{if } d \leq \deg \Lambda(x) + i(k - 1) \text{ return } \Lambda(x)
\]

\[
\text{else begin}
\]

\[
\delta := \delta + 1
\]

\[
\text{for } j = 1, \ldots, L \text{ begin } \tau^{(j)} := \tau^{(j)} - 1 \text{ end}
\]

\[
\text{end}
\]

\[
\text{end}
\]

Once such \( \Lambda(x) = \gamma \Lambda_e(x) \) for some nonzero \( \gamma \in F \) has been found, we can complete decoding by

\[
C(x) = \frac{Y(x)\Lambda(x) \mod m(x)}{\Lambda(x)} \quad (7.12)
\]

or by

\[
C(x) = Y(x) \mod \tilde{m}(x) \quad (7.13)
\]

where \( \tilde{m}(x) \triangleq m(x)/\Lambda(x) \), as mentioned in Section 2.5.

### 7.2 Potential Error-Correcting Radius

We now discuss how to determine \( t \) and \( L \). Suppose that \( \deg \Lambda_e(x) \leq t \), and we hope \( \Lambda_e(x) \) to be the solution \( \Lambda(x) \) of the SPI problem with \( \tau^{(i)} = i(k - 1) + t + 1 \). We note from Theorem 5.1 that the solution \( \Lambda(x) \) satisfies

\[
\deg \Lambda(x) \leq \sum_{i=1}^{L} \left( \deg m^{(i)}(x) - \tau^{(i)} \right) \quad (7.14)
\]

\[
= \sum_{i=1}^{L} \left( n - i(k - 1) - t - 1 \right) \quad (7.15)
\]

\[
= Ln - \frac{L(L + 1)}{2}k + \frac{L(L - 1)}{2} - Lt \quad (7.16)
\]
Since we hope \( \Lambda_e(x) \) with \( \deg \Lambda_e(x) \leq t \) to be the solution \( \Lambda(x) \) (and we hope that \( t \) is as large as possible), the condition

\[
t \leq Ln - \frac{L(L+1)}{2}k + \frac{L(L-1)}{2} - Lt \tag{7.17}
\]

thus need to be satisfied. We then obtain

\[
t^{[L]} \triangleq \left\lfloor \frac{2Ln - L(L+1)k + L(L-1)}{2(L+1)} \right\rfloor \tag{7.18}
\]

that is the largest integer of \( t \) that satisfies (7.17).

We can then determine, from the given \( n \) and \( k \), the respective values of \( L \) and \( t \) as follows: choose \( L \) to be the smallest positive integer that maximizes (7.18), and set \( t \) to be the corresponding maximum of (7.18). Such \( t \) in turn determines \( \tau^{(i)} = i(k-1) + t + 1 \) as mentioned above.

Since \( t^{[L]} \) is a function of \( n \) and \( k \), it is also a function of the rate \( \zeta \triangleq k/n \) of the code \( C \).

For \( L = 1 \), we have \( t^{[1]} = \left\lfloor \frac{n-k}{2} \right\rfloor \) for any rate \( \zeta \), which coincides with the basic error correction bound, i.e., decoding up to half the minimal Hamming distance.

However, \( t^{[L]} \) for \( L \geq 2 \) can only be obtained if

\[
L(k - 1) + t^{[L]} + 1 < n \tag{7.19}
\]

is satisfied, which induces the restriction on code rate

\[
\zeta \leq \frac{2}{L(L+1)} + \frac{L^2 - L - 3}{nL(L+1)} \tag{7.20}
\]

e.g., \( t^{[2]} \) can be obtained only when \( \zeta \leq 1/3 \).

Note that \( L \) and \( t \) determined as mentioned above guarantee \( t = t^{[L]} > t^{[L-1]} \). To easily see the relation between \( L \), \( t^{[L]} \), and rate \( \zeta \), it is convenient to define the normalized error correction radius

\[
\theta^{[L]} \triangleq t^{[L]}/n; \tag{7.21}
\]

we then have

\[
\lim_{n \to \infty} \theta^{[L]} = \frac{L}{L + 1} - \frac{L}{2} \zeta \tag{7.22}
\]

which show the asymptotical behavior of (7.18), cf. Figure 7.1.
Figure 7.1: Illustration of (7.22) for $L = 1, 2, \ldots, 5$.

7.3 Conclusion

We have shown that decoding a scheme of low rate Reed-Solomon codes beyond half the minimal distance as in [41] can be reduced directly and naturally to the SPI problem of Chapter 5. The resulting decoding algorithm is new and has the same error-correcting radius as [41] which is based on the MSSRS algorithms [37, 39].
Appendix A

The Number of Monic Irreducible Polynomials

The number of monic irreducible polynomials of any degree over any finite field can be expressed in closed form \[11\]. However, this closed-form expression is not easy to evaluate. Therefore, for the convenience of the reader, we tabulate some of these numbers.

The first table gives the number \(N_i\) of binary irreducible polynomials of degree \(i\):

\[
\begin{array}{c|cccccccc}
 i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
 N_i & 2 & 1 & 2 & 3 & 6 & 9 & 18 & 30 & 56 \\
 S_i & 2 & 4 & 10 & 22 & 52 & 106 & 232 & 472 & 976 \\
\end{array}
\]

The table also gives the number \(S_i \triangleq \sum_{\ell=1}^{i} \ell N_\ell\), which is the maximum degree of \(M_n(x)\) of a polynomial remainder code that uses only irreducible moduli of degree at most \(i\).

The second table gives the number \(N_i\) of monic irreducible polynomials over \(GF(2^j)\) of degree \(i\):

\[
\begin{array}{c|cccccccc}
 i & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
\hline
 N_i & 99 & 186 & 335 & 630 & 1161 & 2182 & 4080 \\
 S_i & 1966 & 4012 & 8032 & 16222 & 32476 & 65206 & 130486 \\
\end{array}
\]

The table also gives the number \(S_i \triangleq \sum_{\ell=1}^{i} \ell N_\ell\), which is the maximum degree of \(M_n(x)\) of a polynomial remainder code that uses only irreducible moduli of degree at most \(i\).
### The Number of Monic Irreducible Polynomials

<table>
<thead>
<tr>
<th></th>
<th>$GF(2^2)$</th>
<th>$GF(2^4)$</th>
<th>$GF(2^6)$</th>
<th>$GF(2^8)$</th>
<th>$GF(2^{10})$</th>
<th>$GF(2^{12})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>1024</td>
<td>4096</td>
</tr>
<tr>
<td>$N_2$</td>
<td>6</td>
<td>120</td>
<td>2016</td>
<td>32640</td>
<td>523776</td>
<td>8386560</td>
</tr>
</tbody>
</table>

E.g., over $GF(2^8)$, there are 256 monic irreducible polynomials of degree 1 and 32640 polynomials of degree 2.
Bibliography


Series in Signal and Information Processing

edited by Hans-Andrea Loeliger


Vol. 5: Markus Erne, Signal Adaptive Audio Coding using Wavelets and Rate Optimization. ISBN 3-89649-625-5


Vol. 7: Heinz Mathis, Nonlinear Functions for Blind Separation and Equalization. ISBN 3-89649-728-6

Vol. 8: Daniel Lippuner, Model-Based Step-Size Control for Adaptive Filters. ISBN 3-89649-755-3

Vol. 9: Ralf Kretzschmar, A Survey of Neural Network Classifiers for Local Wind Prediction. ISBN 3-89649-798-7


Vol. 11: Pascal O. Vontobel, Algebraic Coding for Iterative Decoding. ISBN 3-89649-865-7

Vol. 12: Qun Gao, Fingerprint Verification using Cellular Neural Networks. ISBN 3-89649-894-0


Hartung-Gorre Verlag Konstanz – http://www.hartung-gorre.de