ulm university universität ulm

Decoding Hermitian Codes - An Engineering Approach

DISSERTATION

zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS
(DR.-ING.)

der Fakultät für Ingenieurwissenschaften
und Informatik der Universität Ulm

von

Sabine Kampf

geboren in Mainz

Gutachter:	Prof. Dr.-Ing. Martin Bossert
	Prof. Peter Beelen
Amtierender Dekan:	Prof. Dr.-Ing. Klaus Dietmayer

Ulm, 5.3.2012

Contents

List of Figures

List of Tables

Abstract

This thesis introduces and discusses a new algorithm for solving the key equation for Hermitian codes, that belong to the class of algebraic-geometric (AG) codes. First, the most important concepts of channel coding are recalled and the popular Reed-Solomon (RS) codes are used to illustrate them. The decoding of RS codes with the extended Euclidean algorithm is used to illustrate the basic idea of algebraic decoding and also to motivate the new decoding algorithm. After that, some elementary results from algebraic geometry are given: only those basics that are (directly or indirectly) used in the definition of AG codes are introduced. From the definition of general AG codes, RS codes and Hermitian codes are derived as special cases. An alternative definition of Hermitian codes that uses almost no algebraic geometry is also given.

After the introduction of the codes, the key equation for Hermitian codes is presented. For a limited error weight, there is a unique solution of minimal degree to the key equation, and the error pattern can be reconstructed from this solution. An algorithm that finds this solution of minimal degree is given. Unfortunately, this algorithm is not capable of decoding all error patterns with weight up to half the minimum distance - a bound up to which unique decoding is guaranteed by the properties of general linear codes that include Hermitian codes. An extension that achieves this decoding radius is discussed afterwards, and the complexity of both the algorithm and its extension are estimated.

A modification of the algorithm that allows to find a basis for decoding beyond half the minimum distance is given in the last chapter. However, decoding up to this increased radius without (significantly) increasing the complexity is not always possible. A bound on the error weight that allows such decoding for interleaved Hermitian codes with high probability is derived, as well as the probability that decoding fails. Finally, the idea of virtual extension to an interleaved code is described. This principle works only for codes with low rates, and therefore the rate bound is given.

Chapter 1

Introduction

The main purpose of channel coding is to allow the correction (or at least detection) of errors that occurred during the transmission of data. This can be both the transmission over a spacial distance - this case corresponds to what is commonly referred to as transmission - or transmission over time, e.g. by storing data on a disk and reading it later. However, there is one problem: if all possible symbol patterns are allowed, there is no way to find out if an error occurred or even correct that error. Hence it is necessary to limit the allowed patterns, or - in other words - transmit redundant data.

As an example, consider the English language and the transmission of single words: not every sequence of letters is an allowed word, but the set of allowed sequences is given by a dictionary, the code. Sometimes, it is possible to detect or even correct errors: if the word "redundunt" is received, one can tell that an error must have occurred during transmission because it is not a word in a dictionary, and that "redundant" is the word that had most probably been sent. But not all errors can be corrected: the word "house" can be transformed into the word "mouse" or "horse" by changing only a single letter and these are words in the dictionary, too. In this case, not even the detection of the error is possible.

One of the main tasks of channel coding is to find good codes, i.e., sets of patterns that allow the correction of many errors while at the same time requiring only few additional symbols to be transmitted. More specifically, one is usually not interested in the maximum number of errors that can be corrected, but in the number of errors for which correction is guaranteed or can at least be performed with high probability. Clearly, language is a bad code because it is not hard to find more pairs of words like "house–mouse" that differ in only one letter, so some errors that change only a single letter cannot be corrected. The other main task is to find algorithms that perform the correction in an efficient way. These two tasks are often contradicting. As an example consider a simple repetition code, where each data bit is transmitted n times. The decision at the receiver is to simply choose the bit (0 or 1) which constitutes the majority of the received bits. To be able to correct 5 errors it is hence necessary to transmit 11 bits for one data bit. Such a redundancy is not acceptable in practical applications.

A theoretical solution to the first problem had been derived by Claude E. Shannon in his famous work "A Mathematical Theory of Communication"[1] in 1948 [Sha48]: given the channel conditions, Shannon derived the exact amount of additional information that needs to be transmitted so that all errors can be corrected. But his proof is nonconstructive, so a lot of engineers and mathematicians have been trying to develop codes that come as close as possible to this bound. So far, it has not been reached and the search continues. One reason is that Shannons proof assumes an infinite number of transmitted symbols: naturally, no practical application will ever transmit that many symbols, and the restrictions of practical channels often limit this number even further. Nevertheless Shannons result implies that it is better to transmit the data in one big block than to split it into a lot of small blocks.

The codes we consider in this thesis belong to the large group of *block codes*, where blocks of information symbols, the information words, are mapped to longer blocks of code symbols, the codewords. The information and code symbols are not necessarily from the same set, but both the information words and codewords have a fixed length. There exist a lot of different classes of block codes that differ in various aspects such as the set from which the information and code symbols are chosen and the decoding methods that can be applied (more details on possible classifications can be found in the next section). For a fair comparison, the complexity of a decoding algorithm for block codes is usually measured relative to the code length, i.e., the number of symbols in one codeword. So far, no algorithms with linear (or even lower) complexity exist for any nontrivial code. The transmission in large blocks hence comes at the cost of increased decoding complexity, and a tradeoff between decoding performance and complexity has to be found.

In this thesis, we use only block codes that can be decoded with algebraic methods. Probably the most widely employed representative of this kind of code are the Reed-Solomon (RS) codes named after their inventors [RS60]: since their first description in 1960, they have been used in a wide variety of applications including CDs, DVDs and deep-space data transmission. Their description is very simple but powerful, and efficient decoding algorithms for these codes were soon found, e.g. the still famous algorithm based on shift-register synthesis found by Berlekamp [Ber68] and Massey [Mas69]. Alternative code descriptions led to other decoding algorithms, e.g. the Sugiyama algorithm [SKHN75] that deploys the extended Euclidean algorithm or the Welch-Berlekamp algorithm [WB86] that uses polynomial interpolation. Despite the differences between these algorithms, the error correction capability is the same for them all.

During the last twenty years, research on RS codes had the main goal to improve these capabilities. Some of the proposed methods use side information about the error (e.g. [Sor93], [Köt96], [KB10], [SSBZ10]), and while these algorithms have the same complexity as the original algorithms they require different receiver structures due to the necessity of side information. Other decoders work without such side information, but they often suffer from severe rate restrictions (e.g. [Sud97], [SSB06]) or no rate restrictions that can be achieved at the cost of increased complexity (e.g. [GS99]).

One big disadvantage of RS codes is that their code length is limited to

[1] Because of the huge impact of this work, it was later republished under the title "THE Mathematical Theory of Communication".

the size of the underlying alphabet, i.e., increasing the code length requires a larger field that complicates the basic operations. Soon, efforts were made to find longer codes over the same base field. One such class of codes are the Hermitian codes for which the main results of this thesis are derived. While it is possible to describe these codes as a generalization of RS codes, they were first found as a special case of the so-called algebraic-geometric (AG) codes. These codes had first been described by Goppa at the beginning of the 1980s (the main paper is [Gop83], but several preliminary papers on the same topics had been published before), and in the following years some more comprehensible introductory works (e.g. [LG88], [Lin90], [HLP98]) were published. In the late 1980s and throughout the 1990s, a lot of decoding algorithms for either specific or general AG codes were introduced - a survey paper by Høholdt and Pellikaan published in 1995 [HP95] lists 112 references - not all of them actually present decoding algorithms, but one should get an idea of how "hot" the topic was.

But even today, more than 15 years later, these codes are not yet used in any practical applications. One reason seems to be that understanding most of the algorithms requires a well-developed background in algebraic geometry that most engineers do not have. Probably in an attempt to change this, Høholdt et al. introduced an alternative description of Hermitian codes in [HLP98] and [JH04] that uses not more algebra than is necessary to understand RS codes. However, the description in [HLP98] is rather unhandy as it includes a general introduction to so-called order functions and the simplified definition of Hermitian codes is left as an example. On the other hand, the chapter in [JH04] is very short, and the connection of their description to the usual notation in other works is not given, making it almost impossible to understand other works about Hermitian codes by only reading the latter book. One of the aims of this thesis is to help bridging this gap by showing how the alternative definition relates to the definition of general AG codes, but without further justifying or generalizing this alternative definition.

About This Thesis

In this thesis, we present all results as simple and comprehensible as possible, but in a way that still allows to see the bigger framework. For this purpose, we give two definitions of Hermitian codes: the first is just a special case of general AG codes, whereas the second is the specific definition adopted from [JH04]. The relation between those two definitions is also given. The description of the algorithm and its extensions in Chapters 5 and 6 is done in a way that it can be understood and implemented using the specific definition, but sometimes algebraic geometry is necessary to understand the proofs. Yet also the definition of general AG codes we give here is not the most general definition that is possible: two types of AG codes are distinguished, but we only present the conceptually simpler type. This is no large drawback because all AG codes can be represented as either type of code (cf. [LG88]).

The thesis is organized as follows: in the next chapter, we shortly recall the basic concepts of channel coding and decoding. In the first section, block codes and their properties and parameters are defined. Few explanations are given because the main purpose of this section is to fix the notations used throughout the thesis. In Section 2.2 general concepts of decoding are introduced, and a specific decoding algorithm for Reed-Solomon codes is given in Section 2.3. We

chose to present decoding with the extended Euclidean algorithm because it is computationally similar to the decoding algorithm presented in Chapter 5.

In Chapter 3 we give a short introduction to the basics of algebraic geometry, where the topics are reduced to those absolutely necessary for the definition of AG codes. Further, many of the definitions are not given in their most general form, but are restricted to the special case needed in the definition of AG codes. Proofs are given only if they are short and illustrative, otherwise the statement of the theorem is illustrated with an example. The design of the examples was always done with the decoding algorithm in mind, e.g. the underlying curve is usually a Hermitian curve because (with a single exception) only codes on Hermitian curves are investigated. Specifically, we introduce the affine and projective line and plane in Section 3.1, and plane algebraic curves in Section 3.2. Both affine and projective spaces and curves also exist in higher dimensions, this is one example where the given definition is restricted to a special case. In Section 3.3, rational functions and divisors are defined, and the calculation of the divisor of a function is illustrated. These divisors are essential for the definition of Riemann-Roch spaces in Section 3.4.

All the basics introduced are then used in Chapter 4 to define algebraic-geometric codes and calculate or estimate their parameters. To justify the use of Reed-Solomon codes in the examples in Chapter 2 it is shown how these codes can be obtained as a subclass of AG codes. Because the decoding algorithm is described for Hermitian codes, another subclass of AG codes, they are given in Section 4.3, along with several specific properties. A separate section is dedicated to the simpler description introduced by Justesen and Høholdt, but in addition to the definitions we also give the relations to the results from the previous sections to illustrate that the definitions do make sense.

We state the decoding problem for Hermitian codes as a key equation in Chapter 5. Several forms of the key equation exist, but we give one that comes close to the key equation for RS codes introduced in Section 2.3 by stating it in terms of polynomials. The core of the decoding algorithm - a division procedure for bivariate polynomials - and the basic decoding algorithm are given next. A proof of correctness of this algorithm follows in Section 5.4. The basic algorithm does not achieve the full correction capabilities of a linear code, therefore Section 5.5 introduces a first extension closing this gap. The chapter concludes with an estimation of the complexity of this algorithm.

Finally, Chapter 6 extends the algorithm even further: if one is willing to give up the certainty of having a unique solution to the decoding problem the decoding capabilities can be further increased and the basic algorithm can be used to obtain a basis for all solutions. The size of this basis is given, and we shortly discuss why this basis should not be used without further information about the error. In Section 6.2 we derive an upper bound on the number of correctable errors if several error words with errors in the same positions are available. This result yields an upper bound on the code rate of a non-interleaved Hermitian code for which an increase in the decoding radius is achieved by virtually extending the received word into an interleaved code at the receiver. This upper bound is given in Section 6.3. The conclusions and a short outlook on possible further problems are given in Chapter 7.

New contributions of the author are found in Chapters 5 and 6. Namely, they are the division decoding algorithm and its extension for decoding up to half the minimum distance in Sections 5.3 and 5.5, which provide an alternative

to existing algorithms. Another new result is the simple method for the decoding of interleaved Hermitian codes presented in Section 6.2, and the derivation of the decoding radius that can be achieved with this simple algorithm. This radius is larger than the radius of previously published algorithms. Section 6.3 shows how this method can be exploited for the decoding of non-interleaved Hermitian codes beyond half the minimum distance. This last algorithm provides an entirely new approach to the problem of decoding beyond half the minimum distance.

Chapter 2

The Basic Idea of Channel Coding

As mentioned in the introduction, the two main tasks of channel coding are finding good codes and efficient decoding algorithms for those codes. A lot of code classes - each with its own advantages and disadvantages - are already known, and so are decoding algorithms for these codes. There exist a lot of textbooks introducing the basics of channel coding, common classes of codes and decoding algorithms for these codes e.g. [MS88], [Bos99], [Bla03] or [JH04], so this chapter covers the basic concepts only to the extent necessary to introduce the notations used throughout the rest of the thesis. Also, only few explicit references are given as the facts can be viewed as common knowledge (and most of the given references are used to demonstrate this).

In this thesis, we cover block codes only. As the name indicates, these codes work by mapping blocks of a fixed number of arbitrary symbols k, the *information words* $\underline{\mathbf{i}}$, to blocks with $n > k$ symbols, the *codewords* $\underline{\mathbf{c}}$. It is known that for a fixed ratio $\frac{k}{n}$, which is called *code rate*, codes that have a larger block length n usually exhibit a better performance. Most practically used block codes are defined over finite fields $GF(q) = GF(p^m)$ with p prime. In this thesis, we concentrate on Reed-Solomon and Hermitian codes, and for these classes the code length is related to the number of elements in the finite field: for Reed-Solomon codes the code length is usually $n = q - 1$, with special extensions the code length can be increased by at most 2. An RS code of longer length can only be obtained by using a larger field, but at the cost of more complex field operations. Another option is the use of codes that have a larger code length over the same finite field like the Hermitian codes introduced later: they have $n = q^{3/2}$. Though the length can be increased even further with other kinds of AG codes, Hermitian codes are expected to be the first kind of AG codes used in applications due to their structural properties.

In the first part of this chapter we discuss the basic parameters of block codes and their encoding. The example of Reed-Solomon (RS) codes is used to illustrate these notions. The second part introduces some basic decoding concepts that mainly differ in the number of correctable error patterns and in the uniqueness of the decoding result. Independently of the concept used, many decoding algorithms rely on syndromes that are introduced afterwards. The calculation of the syndromes is again illustrated with the help of RS codes.

In the last section, a specific decoding algorithm for RS codes is given: the extended Euclidean algorithm is chosen because it motivates the main work of this thesis: the decoding algorithm presented in Chapter 5.

2.1 Block Codes and Their Parameters

Block Codes

Definition 1 (Block Code)**.** *A* block code $\mathcal{C}$ *over a finite field* $GF(q)$ *is a set of* q^k *vectors* $\underline{\mathbf{c}} = (c_0, \ldots, c_{n-1}) \in GF(q)^n$*, the* codewords*, and the length* n *of these vectors is also called the* length *of the code.*

The definition of a code should not be confused with the *encoding* that describes the mapping from the information words to the codewords. Both codes and different encoding rules have their own properties, and we first concentrate on the properties of codes. A code is called *linear* if the linear combination of any two codewords $\underline{\mathbf{c}}, \underline{\mathbf{c}}'$ is again a codeword, i.e., if

$$\underline{\mathbf{c}} + \underline{\mathbf{c}}' = (c_0 + c_0', \ldots, c_{n-1} + c_{n-1}') \in \mathcal{C} \ \forall \ \underline{\mathbf{c}}, \underline{\mathbf{c}}' \in \mathcal{C},$$
$$\alpha \cdot \underline{\mathbf{c}} \in \mathcal{C} \ \forall \ \underline{\mathbf{c}} \in \mathcal{C}, \alpha \in GF(q).$$

In the linear case, the code forms a k-dimensional subspace of $GF(q)^n$, therefore k is called the *dimension* of the code. Another useful property of linear codes is that the all-zero word $(0, \ldots, 0)$ is always in $\mathcal{C}$. The third characterizing parameter of a code is its *minimum distance*. Theoretically, any metric can be used to measure this distance, but the most common choice in coding theory is the Hamming metric:

Definition 2 (Hamming weight and Hamming distance)**.** *The Hamming weight of a vector* $\underline{\mathbf{c}} \in GF(q)^n$ *is the number of its nonzero elements:*

$$w_H(\underline{\mathbf{c}}) = |\{c_i \neq 0 | i = 0, \ldots, n-1\}|.$$

The Hamming distance between two vectors $\underline{\mathbf{c}}, \underline{\mathbf{c}}' \in GF(q)^n$ *is the number of unequal elements. It is equal to the weight of the difference vector* $\underline{\mathbf{c}} - \underline{\mathbf{c}}'$*:*

$$d_H(\underline{\mathbf{c}}, \underline{\mathbf{c}}') = |\{c_i \neq c_i' | i = 0, \ldots, n-1\}| = w_H(\underline{\mathbf{c}} - \underline{\mathbf{c}}').$$

The minimum distance d of a code is the minimum distance between any two different codewords of this code:

$$d = \min\{d_H(\underline{\mathbf{c}}, \underline{\mathbf{c}}') | \underline{\mathbf{c}}, \underline{\mathbf{c}}' \in \mathcal{C}, \underline{\mathbf{c}} \neq \underline{\mathbf{c}}'\}.$$

For a linear code, this number is equal to the minimum weight of any nonzero codeword:

$$d = \min\{w_H(\underline{\mathbf{c}}) | \underline{\mathbf{c}} \in \mathcal{C}, \underline{\mathbf{c}} \neq (0, \ldots, 0)\}.$$

Because these parameters characterize a code and its decoding capabilities, there exists a standard notation that gives all parameters in a compact form:

$$\mathcal{C}(q; n, k, d)$$

is a q-ary code with length n, dimension k and minimum distance d. When the field $GF(q)$ is clear from the context the code is often just denoted as $\mathcal{C}(n, k, d)$.

Sometimes, for example in the case of Reed-Solomon (RS) codes, there is an exact relation between the code parameters and then it is even possible to drop d from the standard notation, e.g. RS codes are often denoted as $\mathcal{RS}(n,k)$ codes. For RS codes, this relation of n, k and d is given towards the end of this section.

As mentioned in Chapter 1 we search for codes that have a large dimension k but can correct many errors. It is not surprising that the correction capabilities are related to the minimum distance d of a code (more details on correction capabilities of a code can be found in Section 2.2), and so we search for codes that have a large minimum distance d given n and k. But d can not be made arbitrarily large, an upper bound is given by the following lemma:

Lemma 1 (Singleton Bound)**.** *For a block code $\mathcal{C}(n,k,d)$, the minimum distance is upper bounded by*

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

Proving this lemma for linear codes is straightforward once the encoding of codes has been treated, so it is deferred to the end of the next part.

Encoding of Block Codes

The encoding is a mapping from the information words $\underline{\mathbf{i}} = (i_0, \ldots, i_{k-1})$ to the codewords $\underline{\mathbf{c}}$. Theoretically any bijective mapping $\psi : GF(q)^k \to \mathcal{C}$ is allowed because the error correction capabilities depend on the code only, but not on the encoding. However, at the receiver side it is often necessary to perform the inverse mapping[2], so one usually chooses a mapping that has a some inherent structure and hence allows efficient decoding. For linear codes, one often chooses a linear mapping, i.e.

$$\underline{\mathbf{i}} \mapsto \underline{\mathbf{c}},\ \underline{\mathbf{i}}' \mapsto \underline{\mathbf{c}}' \ \Rightarrow\ \alpha\underline{\mathbf{i}} + \beta\underline{\mathbf{i}}' \mapsto \alpha\underline{\mathbf{c}} + \beta\underline{\mathbf{c}}' \ \forall\ \underline{\mathbf{i}}, \underline{\mathbf{i}}' \in GF(q)^k,\ \alpha, \beta \in GF(q).$$

Such a linear mapping can be performed by the multiplication of the information word $\underline{\mathbf{i}}$ with a *generator matrix* $\underline{\mathbf{G}}$, i.e.

$$\underline{\mathbf{c}} = \underline{\mathbf{i}} \cdot \underline{\mathbf{G}},$$

where $\underline{\mathbf{i}}$ is a row vector of length k and the matrix $\underline{\mathbf{G}}$ a $k \times n$ matrix of full rank, each row of $\underline{\mathbf{G}}$ being a codeword of $\mathcal{C}$. Any elementary row operations do not decrease the rank of $\underline{\mathbf{G}}$, so they do not alter the code but only the encoding. This fact gives rise to a special form of $\underline{\mathbf{G}}$ that provides the so-called *systematic* encoding. Systematic encoding means that each information symbol i_i appears as a certain code symbol c_j. It is not necessary that the symbols appear in one block or in the same order as in the information word, but a common choice is that the information symbols appear in the same order as the first k symbols of the codeword. For this choice, the generator matrix takes the form

$$\underline{\mathbf{G}} = \left(\underline{\mathbf{I}} \,|\underline{\mathbf{G}}'\right) \tag{2.1}$$

[2]Often those decoding algorithms that immediately return an information word and hence do not require the inverse mapping only work if an encoding with a special inherent structure was used.

where $\underline{\mathbf{I}}$ is a $k \times k$ identity matrix and $\underline{\mathbf{G}}'$ can be any $k \times (n-k)$ matrix. This splitting of the codeword into an information and a redundancy part makes decoding even simpler because the information word can immediately be extracted from the reconstructed codeword $\underline{\hat{\mathbf{c}}}$. Because $\underline{\mathbf{G}}$ has to be a matrix of full rank, any matrix can be transformed to the form (2.1). This means that any code can be encoded in a systematic way[3].

Now we give the proof of Lemma 1 for linear codes: Consider an information word that has only one nonzero symbol. With a systematic encoding, there is only one nonzero symbol among the first k symbols of the codeword. All other $n-k$ symbols may be nonzero, so the total number of nonzero symbols of such a codeword is upper bounded by $n-k+1$, and Lemma 1 follows immediately.

An Example: Reed-Solomon Codes

Reed-Solomon codes can be defined over any finite field. There exist several definitions for RS codes, and we give the one that comes closest to the definition for AG codes given later.

Definition 3 (Reed-Solomon Codes)**.** *Consider the finite field*

$$GF(q) = \{0, \alpha^0 = 1, \alpha, \ldots, \alpha^{q-2}\}.$$

A (primitive) Reed-Solomon code $\mathcal{RS}(n,k,d)$, or $\mathcal{RS}(n,k)$, over $GF(q)$ of length $n = q-1$ and dimension k consists of all codewords

$$\underline{\mathbf{c}} = \left(f(1), f(\alpha), \ldots, f(\alpha^{q-2})\right),$$

where $f(x)$ is a polynomial with coefficients from $GF(q)$ and $\deg(f) \leq k-1$.

It is also possible to construct RS codes with $n < q-1$, the non-primitive RS codes, but later on Hermitian codes are introduced as a possibility to obtain codes with greater length over the same field, so it appears futile to further discuss shorter codes. Extended RS codes can achieve code lengths $n = q$ and $n = q+1$, but this is significantly smaller than the gain in code lengths achieved by using a different code class.

The given definition already suggests to use the following encoding: the elements of an information word $\underline{\mathbf{i}} = (i_0, \ldots, i_{k-1})$ are mapped to the coefficients of the polynomial

$$f(x) = \sum_{j=0}^{k-1} i_j x^j.$$

The codeword is then obtained by evaluating $f(x)$ at all nonzero elements of $GF(q)$. Note that with the implicitly given indexing the code symbols are $c_i = f(\alpha^i)$. Because of the linearity of polynomial evaluation, the definition

[3] In addition to row operations, obtaining this form might require to swap some columns of $\underline{\mathbf{G}}$. But (as mentioned) this does not change the fact that the encoding is still systematic.

directly yields the generator matrix of the code:

$$\underline{\mathbf{G}} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{q-2} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(q-2)} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(q-2)} \\ \vdots & & \vdots & & \vdots \\ 1 & \alpha^{k-1} & \alpha^{(k-1)\cdot 2} & \dots & \alpha^{(k-1)(q-2)} \end{pmatrix}. \tag{2.2}$$

This matrix is a Vandermonde matrix, and this kind of matrices is known to have full rank [LN96]. As mentioned, the length and dimension of an RS code are directly related to the minimum distance of this code. To find this relation, take a closer look at the polynomial evaluation in the definition. Over any field, a univariate polynomial of degree $k-1$ has at most $k-1$ zeros in that field: the univariate polynomials over a field form a euclidean ring. Therefore, if $f(\alpha) = 0$, then it is possible to write

$$f(x) = (x - \alpha) \cdot \bar{f}(x),$$

and $\deg(f) = \deg(\bar{f}) + 1$. The statement follows by induction and the fact that a (nonzero) constant function cannot be zero for any x. Consequently, each codeword has at least $n - (k-1)$ nonzero elements, so $d \geq n - k + 1$. On the other hand, the Singleton bound gives $d \leq n - k + 1$, so it follows that for Reed-Solomon codes $d = n - k + 1$ must hold. A code whose parameters fulfill the Singleton bound with equality is called *maximum distance separable* (MDS), because each codeword can be uniquely reconstructed if any set of k positions is known.

The definition for RS codes given here is one that already includes the encoding rule. By a more general definition, an RS code is obtained if the polynomials $f(x)$ used in the evaluation have nonzero coefficients in at most k consecutive positions, i.e., they are of the form

$$f(x) = \sum_{j=k_0}^{k_0+k-1} f_j x^j, \quad 0 \leq k_0 \leq n - k.$$

Further, in a finite field of order $n+1$ the evaluation of a polynomial $f(x)$ is the same as the evaluation of the polynomial $f(x) \mod x^n - 1$, so it is also possible to have k *cyclically* consecutive nonzero positions. This result is summarized in the next definition.

Definition 4 (Reed-Solomon Codes - Part 2)**.** *The codewords of a (primitive) RS code $\mathcal{RS}(n, k, d)$ over the finite field $GF(q)$ are obtained by evaluating at positions $1, \alpha, \dots, \alpha^{q-2}$ all polynomials of the form*

$$f(x) = f'(x)x^{k_0} \mod x^n - 1, \quad 0 \leq k_0 \leq n-1, \deg(f') < k,$$

where $n = q - 1$.

The generator matrix can be obtained in the same way as before. Compared to (2.2), each element in column i is multiplied by $\alpha^{(i-1)k_0}$. Because x^{k_0} is nonzero for all $0 \neq x \in GF(q)$, it follows that the code symbols which are equal to zero depend on $f'(x)$ only, hence the results on the minimum distance given before still hold.

2.2 Decoding Methods for Block Codes

One thing has been neglected so far: even if a good code (i.e., a code with a large minimum distance d given n and k) is found, it is also necessary to be able to efficiently reconstruct the sent codeword if an error occurs. This process is called *decoding*: given an arbitrary vector $\underline{\mathbf{r}} = (r_0, \ldots, r_{n-1}) \in GF(q)^n$, called the *received word*, that is not necessarily a codeword, the goal is to find the codeword or information sequence that was most probably sent. In practical channels, the probability of a symbol error in transmission is much smaller than the probability that a symbol was received correctly. In terms of Hamming weight, this means that an error of small weight is more probable than one of larger weight, and consequently the most probable sent codeword is the one that is closest to the received word in the Hamming metric. A conceptually simple but very inefficient way to achieve this is to create a table that gives a mapping from all possible received sequences to the closest codeword/ information word, so usually one uses more efficient algorithms.

These decoding algorithms can be characterized in several different ways. The algorithms we present here are all algorithms that use the algebraic structure of the codes, i.e., the fact that the codewords were obtained from polynomial evaluations. They use the syndromes of an error (which are defined later) to first reconstruct the closest codeword $\hat{\underline{\mathbf{c}}}$ and then find the corresponding information word. If a good encoding was chosen, the latter is a trivial task so it is never explicitly given. In contrast to this, there also exist algorithms that do not use the algebraic structure but some more general properties of the code e.g. permutation decoding (see [MS88, Sec. 16.9], [Bos99, Sec. 7.3.1], [Bla03, Sec. 8.5]). A very popular decoding algorithm for RS codes is the Welch-Berlekamp algorithm that (in contrast to the algorithms in this thesis) does not require the calculation of syndromes and directly yields the most probable information word sent without first reconstructing the codeword. The only characterization we treat in more detail is based on the correction capabilities of a decoder, because decoders of different types are presented in this thesis.

Characterization by Correction Capabilities

Most algebraic decoding algorithms can decode a subset of all possible received words efficiently, but are not able to give the closest codeword to each received word. Other decoders (especially for non-algebraic codes) are able to efficiently complete the latter task. Depending on which errors can be corrected, a decoder belongs to one of the following groups.

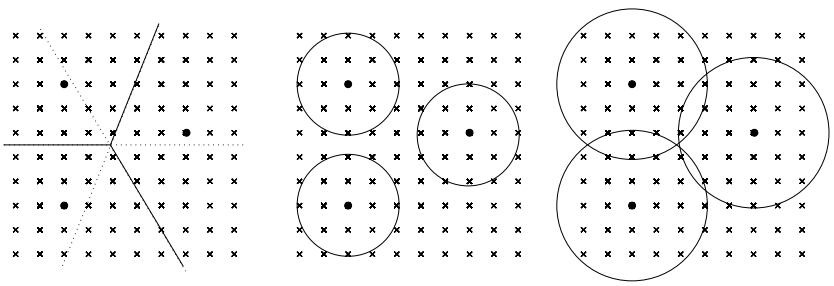
1. Maximum-Likelihood (ML) decoders: a ML decoder always returns the codeword $\underline{\mathbf{c}}$ which is closest to the received word $\underline{\mathbf{r}}$, or, if several codewords are located at the same distance, randomly pick one of these closest codewords. Such a decoder clearly has the best performance in terms of the number of correctable errors. But for most algebraic codes its realization is be very complex due to the shape of the decoding regions, the so-called *Voronoi cells*. This is the reason why no ML decoders are presented in this thesis.

2. Bounded Minimum Distance (BMD) decoders: such decoders only correct those errors whose distance to a codeword is not larger than $t = \lfloor \frac{d-1}{2} \rfloor$,

because for these words there is always a unique closest codeword. If the received word lies outside these *correction spheres* the decoder declares a decoding failure. The advantage of these decoders is that their algorithmic realization is often very simple and if a result exists it is always unique, but for many codes a lot of possible received words do not belong to a correction sphere.

3. Bounded Distance (BD) decoders: these decoders correct those errors whose distance to a codeword is not larger than some $t < d - 1$. Clearly, a larger value of t does not make sense, as then an error might also turn a codeword $\underline{\mathbf{c}}$ into another codeword $\underline{\mathbf{c}}'$, hence it might not be detectable. In many situations, the decoding result is still unique, while the complexity is only slightly increased compared to BMD decoders or even the same. However, the situation when the decoding result is not unique needs to be treated separately. Additionally, there is still a chance that an error cannot be corrected.

For BMD and BD decoders the received words that are decoded into a specific codeword lie in an n-dimensional sphere around the codeword. Therefore the number of correctable errors is often called the *correcting radius* of a decoder. For a BD decoder, one usually assumes that some decoding radius $t > \left\lfloor \frac{d-1}{2} \right\rfloor$ shall be achieved because all errors with weight $t \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ can be corrected by BMD decoders and a decoder with $t < \left\lfloor \frac{d-1}{2} \right\rfloor$ can be obtained by simply rejecting solutions returned by a BMD decoder that correspond to errors of larger weight. In contrast, decoding beyond half the minimum distance usually requires more sophisticated methods.



(a) ML decoding (b) BMD decoding (c) BD decoding

Figure 2.1: Basic decoding principles

Figure 2.1 illustrates the given decoding principles. To make things easier, the sketch is reduced to two dimensions. The codewords are drawn as points and all other possible received words are marked by an "x". In two dimensions, the decoding regions for ML decoding are defined by the perpendicular bisectors of the lines connecting two codewords, for BMD and BD decoding they become circular. One can see that the BMD decoder can correct only a minority of all possible words: only 39 of 100 possible received words lie inside a correction circle. In contrast the BD decoder can correct the majority of all words - only 22 possible received words lie outside a correction circle, and of the other words only 4 lie in more than one correction circle, so 74 of 100 words can be uniquely

mapped to a closest codeword. To see that for practical codes a similar effect arises, consider the following example:

Example 1. Take the $\mathcal{RS}(16, 6)$ code over $GF(17)$. This code has 17^6 codewords, and with decoding radius $t = \left\lfloor \frac{d-1}{2} \right\rfloor = 5$ the BMD correction sphere around one codeword contains

$$1 + 16\binom{16}{1} + 16^2\binom{16}{2} + \cdots + 16^5\binom{16}{5} \approx 4.7 \cdot 10^9$$

possible received words. The total number of vectors $\underline{\mathbf{r}}$ in the correction spheres is hence $1.1 \cdot 10^{17}$. However, theoretically each vector in $GF(17)^{16}$ can be received. There are $4.8 \cdot 10^{19}$ such vectors, so less than every hundredth possible received word can be corrected.

Opposed to this, a BD decoder with $t = 6$ has approximately $1.4 \cdot 10^{11}$ codewords in each correction sphere, but for the total number of received words in all correction spheres one has to take into account that some received words may lie in more than one correction sphere. Due to linearity it is sufficient to consider the all-zero codeword. All error patterns of weight up to 4 lie in only one sphere. An error of weight 5 lies in several correction spheres if it has distance 6 to a codeword of weight 11. The error patterns of weight 6 lying in more than one correction sphere are those having distance 5 or 6 to a codeword of weight 11, or distance 6 to a codeword of weight 12. All error patterns of larger weight are not considered, as they do not lie in the correction sphere of the all-zero word.

First consider the case that 5 errors occurred. For each codeword of weight 11 there are exactly $\binom{11}{5} = 462$ error patterns that have distance 6 to this codeword: the nonzero elements of the codeword must be distributed among the two error words, there may not be overlapping. Because there are 69 888 codewords of this weight, the total number of error patterns of weight 5 lying in two correction spheres is $3.2 \cdot 10^7$ (which is a small fraction of the $4.6 \cdot 10^9$ possible error patterns of weight 5).

The same number is found for errors of weight 6 that have distance 5 to a codeword of weight 11. For error patterns of weight 6 that have distance 6 to a codeword of weight 12, this number is $8 \cdot 10^7$ because the number of codewords of weight 12 is larger. For errors of weight 6 that have distance 6 to a codeword of weight 11, the situation is a little more complicated, but using Formula (29) from [SSB09], one finds that this number is $7.3 \cdot 10^9$, so the number of error patterns lying in more than one decoding sphere is dominated by this term.

But compared with the number of error patterns in one decoding sphere, the uniquely decodable error patterns still constitute the vast majority of cases. Taking all possible codewords into account, their number is approximately $3.4 \cdot 10^{18}$ - this means that now about ten percent of all possible received words can be uniquely decoded, more than ten times as much as a BMD decoder can achieve. •

It can be verified for any code that the fraction of error patterns that are correctable with a BMD decoder gets smaller as the code length increases. In practical scenarios however, a larger fraction of received words can be corrected because errors with small weights occur more often.

Syndrome Decoding

The decoding algorithms presented in this thesis use the *syndromes* of an error as one of the main inputs: these are values that can be calculated from the received word, but depend on the error only. Therefore, the syndromes can be interpreted as a kind of fingerprint of the error. Generally, the vector $\underline{\mathbf{s}}$ of syndromes can be calculated with the help of a check matrix $\underline{\mathbf{H}}$

$$\underline{\mathbf{s}} = \underline{\mathbf{H}} \cdot \underline{\mathbf{r}}^T.$$

The properties of a check matrix are given in the following definition.

Definition 5. *A matrix $\underline{\mathbf{H}}$ is a check matrix for a linear code $\mathcal{C}$ if the following conditions hold:*

- *$\underline{\mathbf{H}}$ is a $(n-k) \times n$ matrix of rank $n-k$.*
- *The syndromes of a codeword are all zero, i.e., $\underline{\mathbf{H}} \cdot \underline{\mathbf{c}}^T = \underline{\mathbf{0}} \, \forall \, \underline{\mathbf{c}} \in \mathcal{C}$.*
- *For a word $\underline{\mathbf{r}} \in GF(q)^n$ that is not a codeword, the product $\underline{\mathbf{H}} \cdot \underline{\mathbf{r}}^T$ contains at least one nonzero element.*
- *d is the minimum distance of $\mathcal{C}$ if and only if any $d-1$ columns of $\underline{\mathbf{H}}$ are linearly independent, and at least one set of d columns is linearly dependent.*

Due to $\underline{\mathbf{r}} = \underline{\mathbf{c}} + \underline{\mathbf{e}}$ and the linearity of matrix multiplications, the syndromes hence depend on the error only. Under certain circumstances, it is then possible to reconstruct the error $\underline{\mathbf{e}}$ from the syndrome $\underline{\mathbf{s}}$, and consequently obtain the originally sent codeword.

Because of the full rank, the parity check matrix also suffices to define the code $\mathcal{C}$. Further, if the code was encoded systematically, it is possible to give a direct relation between generator and check matrix of the code:

$$\underline{\mathbf{G}} = \left(\underline{\mathbf{I}} \, | \underline{\mathbf{G}}'\right) \quad \Leftrightarrow \quad \underline{\mathbf{H}} = \left(-\underline{\mathbf{G}}'^T \, | \underline{\mathbf{I}}'\right),$$

$\underline{\mathbf{I}}'$ is an identity matrix of size $(n-k) \times (n-k)$. With the help of the systematic form, it is always possible to find $\underline{\mathbf{H}}$ if $\underline{\mathbf{G}}$ is given. However, the existence of the parity check matrix is also closely related to the concept of *dual codes*, and this relation sometimes allows to find $\underline{\mathbf{H}}$ more easily.

Definition 6 (Dual Code)**.** *Let $\mathcal{C}$ be a linear code over $GF(q)$. The dual code $\mathcal{C}^\perp$ is a code over $GF(q)$ that consists of all words $\underline{\mathbf{c}}^\perp$ that are orthogonal to all codewords $\underline{\mathbf{c}}$ of the code $\mathcal{C}$, i.e.*

$$\underline{\mathbf{c}} \in \mathcal{C}, \underline{\mathbf{c}}^\perp \in \mathcal{C}^\perp \;\Rightarrow\; \sum_{i=0}^{n-1} c_i c_i^\perp = 0.$$

From the definition of the syndromes it is clear that each row of $\underline{\mathbf{H}}$ is a codeword of the dual code. Closer investigation shows that actually $\underline{\mathbf{H}} = \underline{\mathbf{G}}^\perp$ and $\underline{\mathbf{H}}^\perp = \underline{\mathbf{G}}$. This leads to the relation

$$\mathcal{C}(n,k,d) \leftrightarrow \mathcal{C}^\perp(n, n-k, d^\perp),$$

i.e., the length of a code and its dual code are the same and the dimensions add up to n, but in general there is no direct relation between the minimum distance d of a code and the minimum distance $d^\perp$ of the corresponding dual code. For some classes of codes it is known that the dual of a code belongs to a certain class of code, e.g. the dual code of an RS code is again an RS code. If the generator matrix of this other class of code is known, then $\underline{\mathbf{H}}$ can be easily constructed as the generator matrix of the dual code.

Check Matrix and Syndromes of RS Codes

To find the check matrix of an RS code consider the following theorem:

Theorem 2. *Consider RS codes according to Definition 4. The dual of the code $\mathcal{C} = \mathcal{RS}(n,k)$ with $k_0 = 0$ is $\mathcal{C}^\perp = RS(n, n-k)$ with $k_0 = 1$.*

Proof. The generator matrix of $\mathcal{C}$ was already given in (2.2). To verify the claim, construct the check matrix of an RS code with $k_0 = 1$:

$$\underline{\mathbf{G}}^\perp = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{q-2} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(q-2)} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(q-2)} \\ \vdots & & \vdots & & \vdots \\ 1 & \alpha^{n-k} & \alpha^{(n-k)\cdot 2} & \dots & \alpha^{(n-k)(q-2)} \end{pmatrix}.$$

This matrix is a valid check matrix of $\mathcal{C}$ if the syndrome vector of a codeword is the all-zero vector, i.e., if

$$\underline{\mathbf{s}} = \underline{\mathbf{G}}^\perp \underline{\mathbf{c}}^T = \underline{\mathbf{G}}^\perp \underline{\mathbf{G}}^T \underline{\mathbf{i}}^T = \underline{\mathbf{0}}.$$

Evaluating the matrix product $\underline{\mathbf{G}}^\perp \underline{\mathbf{G}}^T$ results in a matrix of the form

$$\begin{pmatrix} \sum \alpha^i & \sum \alpha^{2i} & \sum \alpha^{3i} & \dots & \sum \alpha^{ki} \\ \sum \alpha^{2i} & \sum \alpha^{3i} & \sum \alpha^{4i} & \dots & \sum \alpha^{(k+1)i} \\ \vdots & & \vdots & & \\ \sum \alpha^{(n-k)i} & \sum \alpha^{(n-k+1)i} & \sum \alpha^{(n-k+2)i} & \dots & \sum \alpha^{(n-1)i} \end{pmatrix},$$

and $i = 0, \dots, n-1$ in each sum. It is a known fact from algebra that over a finite field all these sums evaluate to zero, consequently all syndromes of a codeword are zero independently of $\underline{\mathbf{i}}$. It follows that $\underline{\mathbf{G}}^\perp = \underline{\mathbf{H}}$ for the RS code with $k_0 = 0$. □

As a counterproof, and because this fact clarifies some results in the next section, consider what would happen if $\underline{\mathbf{H}}$ had more than $n-k$ rows: the next two rows in the matrix $\underline{\mathbf{H}}$ would be

$$\begin{pmatrix} 1 & \alpha^{n-k+1} & \alpha^{(n-k+1)\cdot 2} & \dots & \alpha^{(n-k+1)(q-2)} \\ 1 & \alpha^{n-k+2} & \alpha^{(n-k+2)\cdot 2} & \dots & \alpha^{(n-k+2)(q-2)} \end{pmatrix},$$

causing the matrix product $\underline{\mathbf{H}}\,\underline{\mathbf{G}}^T$ to have two additional rows:

$$\begin{pmatrix} \sum \alpha^{(n-k+1)i} & \sum \alpha^{(n-k+2)i} & \dots & \sum \alpha^{(n-1)i} & \sum \alpha^{ni} \\ \sum \alpha^{(n-k+2)i} & \sum \alpha^{(n-k+3)i} & \dots & \sum \alpha^{ni} & \sum \alpha^{(n+1)i} = \sum \alpha^i \end{pmatrix}.$$

These two rows now contain nonzero terms: because $n = q - 1$,

$$\sum_{i=0}^{n-1} \alpha^{ni} = \sum_{i=0}^{n-1} 1 = n = -1 \text{ in } GF(q),$$

so the negative of one coordinate of $\underline{\mathbf{i}}$ appears in $\underline{\mathbf{s}}$. Because this value -1 is moving one position "forward" with each additional row, each of the information coefficients appears in $\underline{\mathbf{s}}$ step by step. Consequently, while the actual check matrix only shows if a received word is a codeword or not, an "extended check matrix"

$$\underline{\widehat{\mathbf{H}}} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{q-2} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(q-2)} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(q-2)} \\ \vdots & & \vdots & & \vdots \\ 1 & \alpha^n & \alpha^{2n} & \dots & \alpha^{n\cdot(q-2)} \end{pmatrix}.$$

having n rows also allows to reconstruct the information word from a given codeword.

2.3 Decoding Reed-Solomon Codes with the Extended Euclidean Algorithm

As mentioned, for many algebraic codes there exist efficient BMD (or BD) decoders, whereas the complexity of ML decoding would be exponential. To give an example of such an efficient BMD decoder and to motivate the work of Chapter 5, we present a method to decode RS codes with the help of the extended Euclidean algorithm (EEA) for polynomials. This decoding method was first introduced in 1975 by Sugiyama et al. [SKHN75], and is presented as a standard decoding algorithm in most textbooks on coding theory nowadays (e.g. [MS88, Sec. 12.9], [Bos99, Sec. 3.2.3], [Bla03, Sec. 7.10]). In all these books, the authors write about "decoding with the EEA", but sometimes the algorithm is also referred to as Sugiyama algorithm to emphasize the necessary modifications introduced in [SKHN75] that make decoding possible.

First, we shortly recall the extended Euclidean algorithm for polynomials. In the next part of this section, we derive the syndrome polynomial of an RS code which is an essential input to the decoding algorithm. After that we give a short motivation for the *key equation* whose solution is the key to decoding RS codes. From the form of the key equation it should become clear why the EEA can be used to decode RS codes. The section is concluded with a small example.

The Extended Euclidean Algorithm

The original version of the Euclidean algorithm is more than 2000 years old and calculates the greatest common divisor of two integers, and the EEA also shows how the greatest common divisor can be obtained as a linear combination of the inputs. Much later, but still long ago, mathematicians found that both algorithms also work for univariate polynomials[4]. The input to the EEA then

[4] More generally, using the mathematical terminology, a Euclidean type algorithm can be performed with two elements from any Euclidean domain.

are two polynomials $r_{-1}(x)$ and $r_0(x)$, not both zero, where w.l.o.g. $\deg(r_0) \leq \deg(r_{-1})$, and the EEA recursively calculates the series of polynomials $r_i(x)$ and $q_i(x)$, where

$$r_i(x) = r_{i-2}(x) - q_i(x)r_{i-1}(x),\ \deg(r_i) < \deg(r_{i-1}). \tag{2.3}$$

If $r_i(x) = 0$, the algorithm stops and $r_{i-1}(x)$ is the greatest common divisor of the two input polynomials. If all coefficients of $r_i(x), q_i(x)$ are from a (not necessarily finite) field this representation is unique, so r_i and q_i can be determined as the remainder and quotient of a polynomial division. Further, the EEA calculates two series of polynomials

$$\begin{aligned} u_i(x) &= u_{i-2}(x) - q_i(x)u_{i-1}(x), \\ v_i(x) &= v_{i-2}(x) - q_i(x)v_{i-1}(x), \end{aligned}$$

with the initializations $u_{-1}(x) = v_0(x) = 0, u_0(x) = v_{-1}(x) = 1$, and the polynomials $q_i(x)$ obtained from (2.3). These polynomials then fulfill the relation

$$\begin{aligned} u_i(x) \cdot r_0(x) + v_i(x) \cdot r_{-1}(x) &= r_i(x) \\ \Leftrightarrow u_i(x) \cdot r_0(x) &= r_i(x) \mod r_{-1}(x) \end{aligned} \tag{2.4}$$

Once the key equation has been introduced it should become clear that the EEA can be used for decoding RS codes, because it has the same form as (2.4). This equality also shows that the $v_i(x)$ are not needed for decoding, so they need not be computed, and the degree condition that is part of the key equation reveals that it is not actually the greatest common divisor that needs to be computed but the execution of the extended Euclidean algorithm can be stopped after some iterations.

The Syndrome Polynomial

As seen before, the extended check matrix allows to reconstruct $\underline{\mathbf{i}}$ if a received word is a codeword. To find the syndrome polynomial, use this extended matrix $\underline{\widehat{\mathbf{H}}}$ with n rows: take an error word $\underline{\mathbf{e}}$, calculate the product

$$\underline{E} = \underline{\widehat{\mathbf{H}}}\,\underline{\mathbf{e}}^T,$$

and map the elements of $\underline{E} = (E_1, \ldots, E_n)$ to the coefficients of the polynomial $E(x) = \sum -E_i x^{n-i}$. It can be verified that under this mapping $E(\alpha^i) = e_i$. In the same way, define the polynomial $R(x)$ for the received word $\underline{\mathbf{r}}$. Because of the linearity of these operations

$$R(x) = E(x) + f(x),$$

where $f(x)$ is the original information polynomial. Because $\deg(f) < k$, it is possible to write

$$R(x) = R'(x) + x^k \cdot S(x),$$

where the polynomial $S(x)$ of degree $\deg(S) \leq n - k - 1$ depends on the error only, but not on the sent codeword. This polynomial is the *syndrome polynomial* that is needed as the input to the decoding algorithm described in the next section.

The Key Equation for RS Codes

Most decoding algorithms for RS codes - among them the Sugiyama algorithm - are so-called *locator decoding* methods. The main idea is to split the decoding into two parts: first, the determination of the error positions and then the determination of the error values. It is this splitting that allows decoding with polynomial complexity, whereas finding the entire error in one step requires solving a nonlinear system of equations, cf. e.g. [Bos99], which is a NP-hard problem.

Finding the error positions is done with the help of a so-called *error locator*. The definition of the error locator is according to Figure 2.2: the vector $\underline{\lambda}$ is zero in every position where $\underline{\mathbf{e}}$ is nonzero, and it is nonzero otherwise, i.e. $\lambda_i = 0 \Leftrightarrow e_i \neq 0$. Further, it is possible to find a polynomial $\Lambda(x)$ that is zero

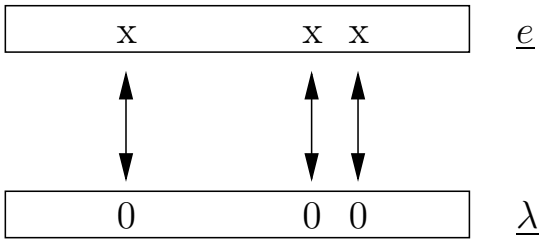


Figure 2.2: Illustration of an error locator in the time domain

at α^i if i is an error position, but at no other positions, i.e.,

$$\Lambda(\alpha^i) = 0 \Leftrightarrow e_i \neq 0. \tag{2.5}$$

Clearly, the polynomial of smallest degree that achieves this has the form

$$\Lambda(x) = c \prod_{i \in \mathcal{I}} (x - \alpha^i), \quad c \in GF(q) \setminus \{0\},$$

and where $\mathcal{I} = \{i | e_i \neq 0\}$ denotes the set of error positions and $t = |\mathcal{I}|$ is the weight of the error word $\underline{\mathbf{e}}$. A polynomial $\Lambda(x)$ that fulfills (2.5) is called an *error locator polynomial*. An error locator is called *proper* if the number of distinct zeros from $GF(q)$ is equal to its degree.

Example 2. The polynomial $\Lambda(x) = x^2 - 3x + 2 = (x-1)(x-2)$ is a proper error locator of degree two. The polynomial $\Lambda_1(x) = x^3 - 4x^2 + 5x - 2 = (x-1)^2(x-2)$ is also an error locator for the same error, but not proper because its degree is larger than its number of zeros ($x = 1$ is counted only once here).

The polynomial $x^6 + x + 1$ has 6 different roots over $GF(2^6)$ but no zeros over $GF(2)$, so it is a proper error locator over $GF(2^6)$, but not over $GF(2)$. •

Of course, $\Lambda(x)$ and the (only partly known) polynomial $E(x)$ fulfill the relation

$$e_i \lambda_i = E(\alpha^i) \cdot \Lambda(\alpha^i) = 0 \ \forall \ i = 0, \ldots, n-1.$$

This is equivalent to the polynomial relation

$$E(x) \cdot \Lambda(x) = \Omega'(x) \prod_{i=0}^{n-1} (x - \alpha^i) = \Omega'(x) \cdot (x^n - 1).$$

The last relation holds in any finite field $GF(q)$ if α is a generator of $GF(q)$ and $n = q - 1$, and these two conditions are fulfilled by the definition of an RS code. Further $\deg(E) \leq n-1$ and $\deg(\Lambda) = t$ by definition, so $\deg(\Omega') < t$. By taking only the known part $S(x)$ of $E(x)$ and limiting the polynomial product to those powers for which the coefficient of $S(x)$ is known, the *key equation* is found:

$$S(x) \cdot \Lambda(x) = \Omega(x) \mod x^{d-1}, \tag{2.6}$$

$$\deg(\Omega) < \deg(\Lambda). \tag{2.7}$$

To make the solution unique, one usually searches for the solution for which $\deg(\Lambda)$ is minimal among all solutions to the key equation. Comparing (2.6) to (2.4), one sees that by choosing the input polynomials $r_{-1}(x) = x^{d-1}$ and $r_0(x) = S(x)$ for the EEA, the pairs $(u_i(x), r_i(x))$ calculated by the EEA fulfill (2.6) for all i. The first pair that further fulfills $\deg(r_i) < \deg(u_i)$, hence (2.7), is the wanted solution of minimal degree, i.e.

$$\Lambda(x) = u_i(x), \Omega(x) = r_i(x) \text{ if } \deg(r_i) < \deg(u_i), \deg(r_{i-1}) \geq \deg(u_{i-1}).$$

Once the error positions have been determined, the remaining task is to find the error values. This can either be done by solving a small linear system of equations with t unknowns (this is demonstrated in upcoming example), or by using the Forney formula that is also found in most textbooks on coding theory (e.g. [Bos99, Sec. 3.2.4], [Bla03, Sec. 7.4], [JH04, Sec. 11.2]). However, the hard part in decoding is the determination of the error positions, whereas the determination of the error values is rather simple once the positions are known. Therefore, in this thesis we often use decoding as a synonym to the determination of the error positions.

Decoding with the EEA - an Example

Consider an $RS(16, 6)$ code. The code is defined over $GF(17)$, and comparison to Definition 3 shows that $\alpha = 3$ is a possible choice for the primitive element of this field. With this choice the parity check matrix of the RS code is

$$\underline{\mathbf{H}} = \begin{pmatrix} 1 & 3 & 9 & 10 & 13 & 5 & 15 & 11 & 16 & 14 & 8 & 7 & 4 & 12 & 2 & 6 \\ 1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 & 1 & 9 & 13 & 15 & 16 & 8 & 4 & 2 \\ 1 & 10 & 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 & 2 & 3 & 13 & 11 & 8 & 12 \\ 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 & 1 & 13 & 16 & 4 \\ 1 & 5 & 8 & 6 & 13 & 14 & 2 & 10 & 16 & 12 & 9 & 11 & 4 & 3 & 15 & 7 \\ 1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 & 1 & 15 & 4 & 9 & 16 & 2 & 13 & 8 \\ 1 & 11 & 2 & 5 & 4 & 10 & 8 & 3 & 16 & 6 & 15 & 12 & 13 & 7 & 9 & 14 \\ 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 & 1 & 16 \\ 1 & 14 & 9 & 7 & 13 & 12 & 15 & 6 & 16 & 3 & 8 & 10 & 4 & 5 & 2 & 11 \\ 1 & 8 & 13 & 2 & 16 & 9 & 4 & 15 & 1 & 8 & 13 & 2 & 16 & 9 & 4 & 15 \end{pmatrix}.$$

Assume that the error $\underline{\mathbf{e}} = (0, 1, 0, 0, \ldots, 0, 1, 0, 0)$ occurred during transmission, i.e., $e_1 = e_{13} = 1$ and $\mathcal{I} = \{1, 13\}$. The syndrome vector $\underline{\mathbf{s}}$ belonging to this error is

$$\underline{\mathbf{s}}^T = (s_0, \ldots, s_{d-2}) = \underline{\mathbf{H}}\,\underline{\mathbf{e}} = (15, 0, 4, 9, 8, 0, 1, 15, 2, 0),$$

and the corresponding syndrome polynomial is

$$S(x) = \sum -s_i x^{d-2-i} = 2x^9 + 13x^7 + 8x^6 + 9x^5 + 16x^3 + 2x^2 + 15x = r_0(x).$$

The second input to the EEA is the polynomial $r_{-1}(x) = x^{d-1} = x^{10}$. In the first iteration, the EEA calculates

$$\begin{aligned} r_1(x) &= 2x^8 + 13x^7 + 4x^6 + 9x^4 + 16x^3 + x^2, \\ q_1(x) &= 9x = -u_1(x). \end{aligned}$$

Clearly, $\deg(r_1) > \deg(u_1)$ so the decoding is not yet finished. In the second iteration,

$$r_2(x) = 15x, \quad q_2(x) = x + 2, \quad u_2(x) = 9x^2 + x + 1.$$

Now $\deg(r_2) < \deg(u_2)$ so one obtains $\Lambda(x) = 9x^2 + x + 1$ and the decoding algorithm stops. To verify that the result is correct, note that this polynomial can be factorized as

$$\Lambda(x) = 9(x^2 + 2x + 2) = 9(x - 3)(x - 12) = 9(x - \alpha)(x - \alpha^{13}).$$

From the latter representation, the error positions 1 and 13 can be read.

To set up the small system of equations that is used to find the error values, any two rows of $\underline{\mathbf{H}}$ can be used. From the first two rows, the resulting linear system is

$$\begin{aligned} 3e_1 + 12e_{13} &= 15, \\ 9e_1 + 8e_{13} &= 0. \end{aligned}$$

Solving this system of equations for e_1 and e_{13} yields the error values $e_1 = 1$, $e_{13} = 1$, which is the error that was chosen in the example. Because the complexity for solving an arbitrary system of equations is cubic, it is advisable to use the Forney formula that achieves the same result in quadratic complexity - especially for larger numbers of errors the gain gets significant.

Chapter 3

Basics of Algebraic Geometry

In this chapter, we introduce the basics of algebraic geometry that are necessary to understand the definition of algebraic-geometric codes in the next chapter. Throughout the chapter we give the definitions and theorems for general plane curves, but at the same time we simplify matters by restricting definitions to only those cases that we actually use. For example, only lines and planes are defined but no higher-dimensional spaces as we do not consider curves or codes in higher-dimensional spaces. In the examples, we almost always use Hermitian curves because codes on Hermitian curves are the only codes for which the decoding algorithm is given later on.

To understand the concepts presented in this chapter, knowledge about elementary algebra such as finite fields is necessary. Most of the concepts presented here can also be found in other introductory work about AG codes, such as [Lin90] and [HLP98], or more general books such as [CLO92], which is an introduction to computational algebraic geometry. Because almost all concepts presented in this section can be found in any of these sources, we give only few explicit references. To keep this chapter as short as possible we skip some topics presented in most introductory work about AG codes that are not absolutely necessary to describe Hermitian codes and their decoding algorithm. Also, we do not give a proof to all theorems, but rather illustrate them with an example where the proof would be too complex.

This chapter is organized as follows: first, the affine and projective line and plane are introduced. Then both affine and projective plane curves and some of their properties are given. In Section 3.3 valuations and divisors of rational functions are defined. Divisors are a prerequisite for the definition of the Riemann-Roch spaces in Section 3.4.

3.1 Affine and Projective Spaces

Affine and projective spaces of any dimension can be defined over any field. However, since only one- and two-dimensional spaces are used in the work presented in this thesis, we only give (rather informal) definitions for lines and planes over finite fields. It is straightforward to extend the given definitions to higher dimensions and arbitrary fields. Such a general definition can be found

in any book on algebraic geometry, e.g. [CLO92].

Definition 7 (Affine Line and Plane)**.** *The affine line consists of all elements $x_p \in GF(q)$. The affine plane consists of all points P that can be described as pairs (x_P, y_P), where $x_P, y_P \in GF(q)$. The affine plane is also referred to as $GF(q)^2$.*

That is, the affine line or plane corresponds to what is usually known as a line or plane over a finite field. If one is further interested in the behaviour of polynomials or rational functions at infinity, it is advisable to consider a projective space.

Definition 8 (Projective Line and Plane)**.** *The projective line consists of all points P that can be described as pairs $(x_P : y_P)$, $x_P, y_P \in GF(q)$ not both zero, and with the equivalence relation $(x_P : y_P) = (ax_P : ay_P)$ for $a \in GF(q) \setminus \{0\}$. The projective plane consists of all points P that can be described as triples $(x_P : y_P : z_P)$, $x_P, y_P, z_P \in GF(q)$, not all zero, and with the equivalence $(x_P : y_P : z_P) = (ax_P : ay_P : az_P)$ for $a \in GF(q) \setminus \{0\}$.*

Usually, the notation of the projective coordinates is fixed in such a way that the rightmost nonzero coordinate of a point P is 1. This convention allows a standard embedding of the affine plane into the projective plane, namely an affine point (x_P, y_P) is represented as the point $(x_P : y_P : 1)$ on the projective plane. In contrast, those points where $z_P = 0$ are the so-called *points at infinity* that also form a projective line.

Example 3. Let $q = 4 = 2^2$ and $GF(2^2) = \{0, 1, \alpha, \alpha^2 = \alpha + 1\}$. The affine plane $GF(4)^2$ then consists of 16 points:

$$\begin{array}{cccc} (0,0), & (0,1), & (0,\alpha), & (0,\alpha^2), \\ (1,0), & (1,1), & (1,\alpha), & (1,\alpha^2), \\ (\alpha,0), & (\alpha,1), & (\alpha,\alpha), & (\alpha,\alpha^2), \\ (\alpha^2,0), & (\alpha^2,1), & (\alpha^2,\alpha), & (\alpha^2,\alpha^2). \end{array}$$

These 16 points can be embedded in the projective plane by the mapping $(x_P, y_P) \mapsto (x_P : y_P : 1)$. Further the projective plane has some points at infinity: for all those points, $z_P = 0$, but because $(0 : 0 : 0)$ is not a point on the projective plane by definition, either $y_P \neq 0$ or $x_P \neq 0$. If $y_P = 0$ then the convention forces $x_P = 1$. If $y_P \neq 0$ the convention gives $y_P = 1$ and x_P can have any value from $GF(q)$. Hence the points at infinity are

$$(1:0:0),\ (0:1:0),\ (1:1:0),\ (\alpha:1:0),\ (\alpha^2:1:0). \qquad \bullet$$

The definition of projective coordinates also has an impact on the definition of polynomials: on the affine plane, bivariate polynomials are used, and they have the usual form

$$f(x,y) = \sum f_{a,b} x^a y^b. \tag{3.1}$$

On the projective plane trivariate polynomials have to be considered, but these have a special form: only *homogeneous* trivariate polynomials are used. A homogeneous polynomial $f(x, y, z)$ with $\deg(f) = i \geq \max\{a + b | f_{a,b} \neq 0\}$ has the form

$$f(x,y,z) = \sum f_{a,b} x^a y^b z^{i-a-b}. \tag{3.2}$$

In order to restrict a function given on the projective plane to the affine plane, it is enough to simply drop all occurrences of z - this is possible because all points that are both on the affine and the projective plane have $z_P = 1$ due to the chosen normalization. In order to extend a function on the affine plane to the projective plane, one uses $i = \max\{a + b | f_{a,b} \neq 0\}$ and then multiplies each summand $f_{a,b}x^a y^b$ of $f(x, y)$ by z^{i-a-b} to form the homogeneous polynomial $f(x, y, z)$ of minimal degree.

To denote the evaluation of $f(x, y, z)$ at a point $P = (x_P : y_P : z_P)$ one can write $f(x_P, y_P, z_P)$. However it is much shorter to write $f(P)$, which is why we use that latter option from now on. Note that due to the equivalence $(x_P : y_P : z_P) = (ax_P : ay_P : az_P)$ with $a \in GF(q)\backslash\{0\}$ the value is only defined up to a nonzero scalar. In contrast to this, the value of a homogeneous rational function is well defined also over the projective plane, so even if a polynomial $f(x, y, z)$ is given, it is usually interpreted as the rational function

$$\frac{f(x, y, z)}{z^i}, \text{ where } i = \deg(f).$$

Because the short notation $f(P)$ can also be used for affine polynomials and points, it is required to conclude from the context whether affine or projective coordinates are used.

3.2 Algebraic Plane Curves

For the purposes considered in this thesis, an algebraic curve can be seen as a set of points that fulfills a polynomial equation (with some restrictions on the polynomial). More general definitions are possible, but require a more intensive study of the concepts of algebraic geometry. Further, one may consider curves embedded into a projective space of arbitrary dimension, but we restrict ourselves to *plane* curves, i.e., curves embedded in a two-dimensional space, and do not explicitly mention this fact any more. While all given results hold for plane curves, they might not be true in higher dimensions. A plane curve is defined by a bivariate or homogeneous trivariate polynomial, depending on whether affine or projective coordinates are used.

Definition 9 (Algebraic Plane Curves)**.** *An affine curve $\mathcal{X}$ over $GF(q)$ is the set of affine points $(x_P, y_P) \in GF(q)^2$ that fulfills the* defining equation

$$h(x, y) = 0, \quad h(x, y) \in GF(q)[x, y].$$

The polynomial $h(x, y)$ is called the defining polynomial *and needs to be absolutely irreducible (i.e., irreducible not only over $GF(q)$ but also over all extension fields). Similarly, a projective curve $\mathcal{X}$ over $GF(q)$ is the set of projective points $(x_P : y_P : z_P)$ that fulfills the defining equation*

$$h(x, y, z) = 0,$$

where the defining polynomial $h(x, y, z)$ is a homogeneous absolutely irreducible polynomial in $GF(q)[x, y, z]$.

Note that the same notations are used for both affine and projective curves. However, any affine curve gives rise to a unique projective curve that is obtained

by homogenizing the defining polynomial. For these two curves, the only difference is that the projective curve may have some additional points at infinity which in fact are already determined by the affine curve. Whenever the behaviour at infinity is not important, it is sufficient to consider the affine curve, otherwise the projective curve should be used. Codes are often defined with the help of the points at infinity, but only affine points are used in the encoding and decoding.

Example 4 (Hermitian Curves)**.** The curves that we use most in this work are the *Hermitian curves* in their so-called *Stichtenoth version*. The Hermitian curve over $GF(q^2)$ has the defining polynomial

$$h(x,y) = x^{q+1} - y^q - y \quad \text{or} \quad h(x,y,z) = x^{q+1} - y^q z - y z^q.$$

Note that it is not a restriction of the possible finite fields if one denotes the base field by $GF(q^2)$: assume that the curve is defined over the finite field $GF(Q)$, then

$$\tilde{h}(x,y) = x^{\sqrt{Q}+1} - y^{\sqrt{Q}} - y.$$

An expression of this form is a polynomial if and only if $\sqrt{Q}$ is an integer, so $\tilde{h}(x,y)$ only defines an algebraic curve if Q is square, i.e., $Q = q^2$.

Using the defining polynomial $x^{q+1} - y^q - y$ over another base field, e.g. $GF(q)$, also yields an algebraic curve, but one whose number of $GF(q)$-rational points is significantly smaller than the maximal number of points a curve can have for a given degree of the defining polynomial. •

A requirement in many of the following definitions and theorems is that a certain point on the curve needs to be *nonsingular*.

Definition 10 (Singular Points and Regular Curves)**.** *A point P on a curve $\mathcal{X}$ with defining polynomial $h(x,y,z)$ is called* singular *if all partial derivatives vanish at this point, i.e.,*

$$h_x(P) = h_y(P) = h_z(P) = 0.$$

If a curve has no singular points over the algebraic closure it is called nonsingular*,* regular *or* smooth*.*

Over the field of real numbers, a regular plane curve is one which does not intersect with itself and has no "corners". Over a finite field, unfortunately no descriptive explanation exists. Though it would be possible to exclude singular points in all further considerations, this would make the notation very unhandy, so we choose to treat regular curves only. One of the reasons why we concentrate on Hermitian curves is that they are nonsingular independent of the field over which they had been defined.

Lemma 3. *All Hermitian curves are regular.*

Proof. The defining polynomial of a projective Hermitian curve is given by $h(x,y,z) = x^{q+1} - y^q z - y z^q$, so the partial derivatives and the conditions that they are zero are

$$\begin{aligned} h_x &= (q+1)x^q = x^q \Rightarrow x = 0, \\ h_y &= -q y^{q-1} z - z^q = -z^q \Rightarrow z = 0, \\ h_z &= -y^q - q y z^{q-1} = -y^q \Rightarrow y = 0. \end{aligned}$$

Under these conditions, the only singular point would be $(0 : 0 : 0)$, but by definition this is no point on the projective plane. Hence the curve is regular. □

There are several standard parameters that can be used to describe algebraic curves. The two most important for this thesis are the degree and genus of a curve:

Definition 11 (Properties of Algebraic Curves)**.** *The* degree $d(\mathcal{X})$ *of a curve is the degree of its defining polynomial: for an affine curve with defining polynomial* $h(x, y) = \sum h_{a,b} x^a y^b$ *the degree is*

$$d(\mathcal{X}) = \deg\,(h(x, y)) = \max\{a + b | h_{a,b} \neq 0\},$$

for a projective curve it is the degree of the homogenized polynomial $h(x, y, z)$*. The* genus $g(\mathcal{X})$ *of a regular projective curve, i.e., a curve that is also regular at the points at infinity, can be calculated from* $d(\mathcal{X})$ *by the Plücker formula [Lin90, Thm. 8.1]*

$$g(\mathcal{X}) = \frac{1}{2}(d(\mathcal{X}) - 1)(d(\mathcal{X}) - 2).$$

If the projective curve is regular the Plücker formula can be used for the affine curve, too, whereas nonsingularity in the affine points is not sufficient. Because it is usually clear from the context which curve $\mathcal{X}$ is referred to, we denote the genus by g only. So far, the genus can be simply seen as a parameter that can be calculated. In Section 3.4 it becomes clear why the genus is an important parameter in the description of AG codes.

Example 5. A Hermitian curve $\mathcal{X}$ over $GF(q^2)$ has degree $d(\mathcal{X}) = q + 1$ and genus $g = \frac{1}{2}q(q - 1) = \frac{1}{2}(q^2 - q)$. •

Though all results given so far hold for arbitrary plane curves, we only describe the decoding algorithm for Hermitian codes. Therefore, it seems reasonable to mention a result specific for those curves:

Lemma 4 ([Lin90, Thm. 11.2])**.** *The Hermitian curve over* $GF(q^2)$ *has* q^3 *affine points and a single point at infinity.*

Proof. First we prove that there is a single point at infinity: all points at infinity have $z = 0$. The defining equation hence becomes

$$x^{q+1} = 0 \Leftrightarrow x = 0.$$

Because at least one coordinate must be nonzero if follows that $y \neq 0$, and with the given convention the point at infinity is $P = (0 : 1 : 0)$.

The derivation for the number of affine points is a little more involved. First consider $x = 0$, then the defining equation becomes

$$y^q + y = 0 \Leftrightarrow y = 0 \text{ or } y^{q-1} = 1.$$

Because in $GF(q^2)$ the relation $\alpha^{q^2-1} = 1$ always holds, the last part has $q - 1$ solutions: $1, \alpha^{q+1}, \alpha^{2(q+1)}, \ldots, \alpha^{(q-2)(q+1)}$ with α a generator of $GF(q^2)^*$, so there are q solutions with $x = 0$. If $x = \alpha_i \neq 0$, one has to solve

$$y^q + y = \alpha_i^{q+1}. \tag{3.3}$$

There are q solutions for each of the q^2-1 possible α_i, but it is also necessary to show that all zeros are from $GF(q^2)$. To do so, first note that $\alpha_i^{q+1} \in GF(q)$ because

$$\left(\alpha_i^{q+1}\right)^q = \alpha_i^{q^2+q} = \alpha_i^{q+1}.$$

On the other hand, if $y \in GF(q^2)$ then $y^q + y \in GF(q)$ because

$$(y^q + y)^q = y^{q^2} + y^q = y + y^q. \tag{3.4}$$

Considering the group homomorphism

$$\mathcal{P} : GF(q^2) \to GF(q^2), y \mapsto y^q + y,$$

(3.4) shows that $\mathrm{Im}(\mathcal{P}) = GF(q)$, and elementary algebra yields $|\mathrm{Ker}(\mathcal{P})| = q$, so each of the q solutions to (3.3) must be from $GF(q^2)$. Consequently the overall number of affine points is

$$q(q^2-1) + q = q^3.$$

□

More details on this derivation can be found in the standard literature. As implicitly introduced in the proof, the point at infinity is often denoted by P. We usually refer to the other points as $P_1, \ldots, P_n$.

Example 6. We conclude this section by giving the points on two concrete Hermitian curves. First, let $q = 2$, i.e., $h(x, y) = x^3 - y^2 - y$. The corresponding affine Hermitian curve then has 8 points:

$$\begin{array}{cccc} (0,0) & (0,1) & (1,\alpha) & (1,\alpha^2) \\ (\alpha,\alpha) & (\alpha,\alpha^2) & (\alpha^2,\alpha) & (\alpha^2,\alpha^2) \end{array}$$

The points on a curve can also be visualized in an array [Bla08]. The visualization for the projective curve over $GF(2^2)$ is given in Figure 3.1. Each square stands for a point on the projective plane, where the x-coordinate is given on the horizontal axis and the y-coordinate on the vertical axis. The line at infinity is drawn at the top and denoted by $y = \infty$, the point $(1 : 0 : 0)$ by $x = y = \infty$. Though the latter two notations are not consistent with the coordinates of the points, the main motivation behind it is to make the points at infinity easily identifiable. The points on the curve are marked with black circles.

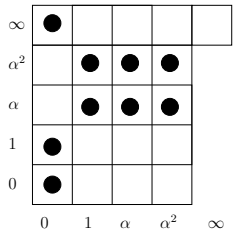


Figure 3.1: Hermitian curve over $GF(2^2)$

For $q = 4$, the curve has already 65 points, so listing them all is not very practical any more. Therefore, we only visualize them in the same way as before.

This is shown in Figure 3.2. This visualization shows that $\mathcal{X}$ has a very regular structure that induces structural properties of the code. This is one of the main reasons why Hermitian codes are expected to yield good performance. •

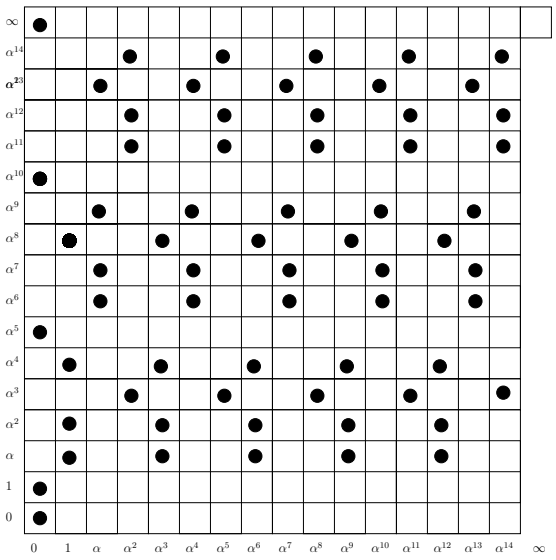


Figure 3.2: Hermitian curve over $GF(2^4)$

3.3 Divisors, Valuations and Rational Functions

When working on algebraic curves, divisors are a useful tool to describe properties of polynomials and rational functions on the curve. However, divisors can be formally defined independent of rational functions. Therefore, we first introduce divisors and give their properties before defining valuations and showing how these can be calculated. In the last part divisors of rational functions are introduced, that keep track of the valuation of a function at all points on a curve.

Divisors

Definition 12 (Divisor)**.** *A divisor D is a formal sum*

$$D = \sum_{P \in \mathcal{X}} n_P \cdot P,$$

where $\mathcal{X}$ is an arbitrary curve, $n_P \in \mathbb{Z}$ and only finitely many n_P are unequal to zero.

Formally, this definition also includes points over any extension field, and at the end of this section Example 12 shows why these points usually cannot be neglected. But when working with Hermitian codes it is usually safe to ignore them because only points over the base field are used in the encoding, and consequently also in the decoding. Because this number of points is finite, the

last constraint becomes insignificant. The information given by a divisor could also be given by a list of pairs (n_P, P), but the notation as a sum gives rise to an intuitive addition of divisors

$$D_1 + D_2 = \sum_{P \in \mathcal{X}} n_{P,1} \cdot P + \sum_{P \in \mathcal{X}} n_{P,2} \cdot P = \sum_{P \in \mathcal{X}} (n_{P,1} + n_{P,2}) \cdot P.$$

This addition is often used, e.g. in Definition 16 in the next section.

Definition 13 (Properties of a Divisor)**.** *The* support *of a divisor is*

$$\text{supp}(D) = \{P | n_P \neq 0\}.$$

The degree *of a divisor is defined as*

$$\deg(D) = \sum_{P \in \mathcal{X}} n_P.$$

If $n_P \geq 0 \, \forall \, P$, a divisor is called effective*. This is denoted by $D \geq 0$.*

While this definition does not restrict the possible origins of the coefficients n_P, the main application is that divisors can be used as a means to keep track of the zeros and poles of rational functions as well as their multiplicities or orders respectively. However, the notion of multiplicities and orders is slightly different for functions on curves, and to be able to calculate them it is first necessary to introduce *valuations*.

Rational Functions on Curves

When working with rational functions on curves, one should keep in mind that any function that is a multiple of the defining polynomial $h(x, y)$ is equal to the all-zero function on the curve. This yields a partitioning of all rational functions into equivalence classes, and each rational function can be represented as

$$g(x, y) = \frac{g_1(x, y)}{g_2(x, y)} = \sum_{i=0}^{d_h - 1} g_i(x) y^i,$$

where $d_h = \deg_y(h)$, hence $d_h = q$ for Hermitian curves, and $g_i(x) = \frac{g_{i1}(x)}{g_{i2}(x)}$ is a rational function in only one variable. Such a rational function induces a unique mapping from the points on the curve to the projective line over the same base field.

Example 7. To illustrate the mentioned mapping, consider the function $g(x, y) = \frac{y}{x}$ and denote the affine points on a Hermitian curve by $P_i = (\alpha_i, \beta_i)$. If $\alpha_i \neq 0$, the value is determined by simply evaluating $g(x, y)$ in P_i:

$$P_i \mapsto g(\alpha_i, \beta_i) = \frac{\beta_i}{\alpha_i}.$$

For $\alpha_i = 0$ and $\beta_i \neq 0$, the mapping is defined as $P_i \mapsto \infty$. For the point $(0, 0)$ and the unique point at infinity a statement is also possible, but the *valuations* have to be used as these give some information about the multiplicities of the respective zeros of $g_1(x, y)$ and $g_2(x, y)$. •

For rational functions in a projective space, it is not sufficient that the numerator and denominator are homogeneous polynomials, but they also need to have the same degree. Otherwise the rational function is not well-defined [Lin90]. This also implies a unique embedding of polynomials into the rational functions - a polynomial of degree i is simply divided by the monomial z^i.

Valuation of Rational Functions

A valuation is a mapping from rational functions to integers with some additional properties. These properties are listed e.g. in [HLP98]. Here, we introduce such a mapping without giving the proof that it actually has the required properties.

Consider a rational function f and fix any point Q on the curve. In this point Q the valuation takes the following values:

1. $f(Q) \neq 0$ and $f(Q) \neq \infty$. In this case, the valuation is $v_Q(f) = 0$.
2. $f(Q) = 0$. Then $v_Q(f) > 0$ and $v_Q(f)$ is the multiplicity of this zero.
3. $f(Q) = \infty$. In this case, $v_Q(f) < 0$ and $-v_Q(f)$ is the pole order.

Clearly, this is not a formal definition of a valuations, instead we concentrate on demonstrating how to calculate them.

Example 8. Consider the function x^2 on the affine line over $\mathbb{R}$. It is well known that this function has a double zero in $P_0 = 0$, so $v_{P_0}(x^2) = 2$. In $P_1 = 1$, $x^2 = 1 \neq 0$ so $v_{P_1}(x^2) = 0$. •

For the given example, one finds the multiplicities with the help of derivatives. When working over finite fields, it is necessary to use formal derivatives to avoid problems arising from the characteristic of the underlying field. Another problem is that the valuation of a rational function on a curve is often not very easy to determine because the notion of derivation is quite different compared with univariate functions. Instead, valuations are usually calculated with the help of so-called *local parameters*[5]. In most introductions to AG codes, more formal definitions are given, but we prefer to give one that directly implies how to find such a local parameter.

Definition 14 (Local Parameter [Giu])**.** *A local parameter t_P is a rational function of the form*

$$t_P = \frac{f_1(x, y, z)}{f_2(x, y, z)},$$

where f_1, f_2 are homogeneous polynomials with $\deg(f_1) = \deg(f_2) = 1$*, $f_1(P) = 0$, $f_2(P) \neq 0$ and f_1 is not a constant multiple of*

$$\tau_P = h_x(P)(x - x_P) + h_y(P)(y - y_P) + h_z(P)(z - z_P).$$

As before, h_x, h_y and h_z are the partial derivatives of the defining polynomial $h(x, y, z)$, so τ_P is the tangent to $\mathcal{X}$ in P. That is, a local parameter is a function that intersects $\mathcal{X}$ in P but is not tangent to it or, in other words, a function that has a simple zero in P. In singular points $\tau_P = 0$ and no local parameter

[5]Other common terms in literature are *uniformizing parameter* or *uniformizing element*.

exists. Although the nonsingularity is never used explicitly, the existence of a local parameter is crucial for the following definitions and theorems leading to the definition of a code, which is why we work with regular curves only.

With the help of the local parameter, each function can be written as

$$f(x,y,z) = t_P^{v_P} \cdot \tilde{f}(x,y,z) \text{ with } \tilde{f}(P) \notin \{0,\infty\}, \tag{3.5}$$

where v_P is the valuation of f in P, so it can be read off directly from this form. Note that this equation is written in projective coordinates. To find the valuation of an affine function, it is always advisable to rewrite the function as a homogeneous rational function and calculate the valuation in projective coordinates. At the affine points, the valuation of a rational function and its homogenized counterpart are always the same. Besides, there exist relations between the valuations of functions that also can be helpful:

$$v_P(fg) = v_P(f) + v_P(g), \tag{3.6}$$
$$v_P(f^{-1}) = -v_P(f), \tag{3.7}$$
$$v_P(f+g) \geq \min\{v_P(f), v_P(g)\}, \tag{3.8}$$

and the last equation is fulfilled with equality if $v_P(f) \neq v_P(g)$.

Example 9. Take the Hermitian curve over $GF(4^2)$ with defining polynomial $h(x,y,z) = x^5+y^4z+yz^4$. The field has characteristic two, so the formal partial derivatives are $h_x = x^4$, $h_y = z^4$ and $h_z = y^4$. Let $P = (0:1:0)$ be the point at infinity. In this point $\tau_P = z$, and a possible local parameter[6] is $t_P = \frac{x}{y}$.

To calculate the multiplicity of the function $\frac{x}{z}$, rewrite it with the help of the defining polynomial:

$$\begin{aligned}\frac{x}{z} &= \frac{x}{x^5/(y^4+yz^3)} = \frac{y^4+yz^3}{x^4} \\ &= \frac{y^4}{x^4} \cdot \left(1+\frac{z^3}{y^3}\right) = \frac{y^4}{x^4} \cdot \frac{y^3+z^3}{y^3} = t_P^{-4} \cdot \tilde{f}(x,y,z).\end{aligned}$$

Because $\tilde{f}(P) = 1$, the valuation $v_P\left(\frac{x}{z}\right) = -4$ can be read off directly. With this result, (3.6) and (3.7), it is now simple to find $v_P\left(\frac{y}{z}\right)$:

$$v_P\left(\frac{y}{z}\right) = v_P\left(\frac{x}{z}\right) + v_P\left(\frac{y}{x}\right) = -5.$$

This result can be extended to the general form

$$v_P\left(\frac{x^a y^b}{z^{a+b}}\right) = -4a - 5b. \qquad \bullet$$

In a similar way, it is possible to calculate the valuations of functions in all points. Two more examples are given in Appendix A. The reason to give the valuation of the functions $\frac{x^a y^b}{z^{a+b}}$ here is that these functions (and their valuations) play an important role in the definition of the Hermitian codes in Section 4.3.

[6] Note that the local parameter is not necessarily unique. In the given example, $\tilde{t}_P = \frac{x+z}{x+y+z}$ is another possible local parameter, but this choice would make it much harder to represent a function in the form (3.5).

Finding a representation as in (3.5) can be very hard, especially if t_P cannot be expressed in an equally simple way as in the previous example. Another form that allows to directly read off the valuation of a function is its representation as a power series in a local parameter

$$f(x,y,z) = \sum_{i \geq v_P} t_P^i c_i, \tag{3.9}$$

where the coefficients $c_i \in GF(q)$ are constants. To find the valuation, it is not necessary to determine the entire power series but instead it suffices to find the smallest power i for which the coefficient c_i is nonzero. Because the determination of the valuation is an essential part of the decoding algorithm (in the determination of the error positions), we illustrate this second option in an example. To obtain a shorter notation, we use the affine curve in the example, but the calculation works in the same way as explained for projective curves before.

Example 10 (Determination of the Valuation with a Power Series)**.** Consider again the affine Hermitian curve over $GF(4^2)$, i.e., $h(x,y) = x^5 - y^4 - y$, and the polynomial $f(x,y) = \alpha^4 y^2 + \alpha^{11} xy + \alpha^8 x^2 + \alpha^{11} y + \alpha^4$. This polynomial has 6 affine zeros on the curve:

$$P_1 = (1, \alpha^8), P_2 = (\alpha^2, \alpha^3), P_3 = (\alpha^2, \alpha^{14}),$$
$$P_4 = (\alpha^3, \alpha^2), P_5 = (\alpha^{12}, \alpha^8), P_6 = (\alpha^{13}, \alpha^{13}).$$

There might also be further zeros over an extension field which we do not consider in this example. We derive the valuations in the points P_2 and P_6. The valuations in the other points can be determined analogously.

First, consider the point $P_2 = (\alpha^2, \alpha^3)$. Using Definition 14 it is easy to verify that $t_{P_2} = x - \alpha^2 = x + \alpha^2$ is a local parameter. Using the relation

$$x^5 - (\alpha^2)^5 = y^4 + y - ((\alpha^3)^4 + \alpha^3),$$

one obtains

$$y - \alpha^3 = y + \alpha^3 = (x + \alpha^2) \cdot \frac{x^4 + \alpha^2 x^3 + \alpha^4 x^2 + \alpha^6 x + \alpha^8}{y^3 + \alpha^3 y^2 + \alpha^6 y + \alpha^7}.$$

This relation allows the following manipulations on $f(x,y)$:

$$\begin{aligned} f(x,y) &= \alpha^4 y^2 + \alpha^{11} xy + \alpha^{11} y + \alpha^8 x^2 + \alpha^4 \\ &= \alpha^4 y^2 + \alpha^{11} y(x + \alpha^2) + \alpha^4 y + \alpha^8 (x + \alpha^2)^2 + \alpha^6 \\ &= \alpha^4 (y + \alpha^{14})(y + \alpha^3) + \alpha^{11} y(x + \alpha^2) + \alpha^8 (x + \alpha^2)^2 \\ &= \alpha^4 (y + \alpha^{14})(x + \alpha^2) \cdot \frac{x^4 + \alpha^2 x^3 + \alpha^4 x^2 + \alpha^6 x + \alpha^8}{y^3 + \alpha^3 y^2 + \alpha^6 y + \alpha^7} \\ &\quad + \alpha^{11} y(x + \alpha^2) + \alpha^8 (x + \alpha^2)^2. \end{aligned}$$

Clearly, the term $\alpha^8 (x + \alpha^2)^2$ has a double zero in P_2, but

$$\alpha^4 (y + \alpha^{14}) \cdot \frac{x^4 + \alpha^2 x^3 + \alpha^4 x^2 + \alpha^6 x + \alpha^8}{y^3 + \alpha^3 y^2 + \alpha^6 y + \alpha^7} + \alpha^{11} y \Big|_{P_2} = \alpha^5,$$

so $f(x,y)$ has a zero of multiplicity 1 in P_2.

Repeating the same steps for $P_6 = (\alpha^{13}, \alpha^{13})$, one finds the local parameter $t_{P_6} = x + \alpha^{13}$ and

$$f(x,y) = (x+\alpha^{13}) \left[\alpha^4 y \frac{x^4 + \alpha^{13}x^3 + \alpha^{11}x^2 + \alpha^9 x + \alpha^7}{y^3 + \alpha^{13}y^2 + \alpha^{11}y + \alpha^7} + \alpha^{11} y \right] + \alpha^8 (x+\alpha^{13})^2,$$

but the term in square brackets evaluates to zero in P_6, so $f(x,y)$ has a zero of multiplicity at least two in P_6. Therefore, we continue by next determining the valuation of the term in square brackets. The denominator of the fraction is unequal to zero, so we multiply with it and find

$$\begin{aligned}
&\alpha^4 y(x^4 + \alpha^{13}x^3 + \alpha^{11}x^2 + \alpha^9 x + \alpha^7) + \alpha^{11} y(y^3 + \alpha^{13}y^2 + \alpha^{11}y + \alpha^7) \\
=&\alpha^4 y(x^4 + \alpha^7) + \alpha^2 y(x^3 + \alpha^{13}x^2 + \alpha^{11}x) + \alpha^{11} y(y^3 + \alpha^{13}y^2 + \alpha^{11}y + \alpha^9 + 1) \\
=&\alpha^4 y(x+\alpha^{13})^4 + \alpha^2 y(x+\alpha^{13})^3 + \alpha^{11} y(y+\alpha^{13})^3.
\end{aligned}$$

From the last equation, it can be seen that the term in brackets has a zero of multiplicity at least three, so the valuation is now dominated by $\alpha^8(x+\alpha^{13})^2$, which means that $f(x,y)$ actually has a zero of multiplicity two in P_6. •

Divisors of Rational Functions

Based on valuations of rational functions, it is now possible to define the previously mentioned divisors of rational functions that keep track of the poles and zeros of a function along with the respective orders or multiplicities.

Definition 15 (Divisor of a Rational Function)**.** *Consider a homogeneous rational function $f(x,y,z)$ on a projective curve. Then the* divisor of f *is*

$$\operatorname{div}(f) = \sum_{P \in \mathcal{X}} v_P(f) P.$$

Note that in this definition it is absolutely necessary to include the points over the extension fields.

Example 11. We continue Example 10. Consider the homogeneous rational function

$$f_h(x,y,z) = \frac{\alpha^4 y^2 + \alpha^{11} xy + \alpha^8 x^2 + \alpha^{11} yz + \alpha^4 z^2}{z^2},$$

which is obtained by homogenizing the polynomial $f(x,y)$. When counting the multiplicities of the zero in P_1, P_3, P_4 and P_5, one finds that $f(x,y,z)$ has a single zero in all these points, and $P_1, \ldots, P_6$ are all rational affine points of $\mathcal{X}$. For the single point P at infinity, it follows from (3.8) and Example 9 that

$$\begin{aligned}
v_P(f_h) &= \min\left\{ v_P\left(\frac{y^2}{z^2}\right), v_P\left(\frac{xy}{z^2}\right), v_P\left(\frac{x^2}{z^2}\right), v_P\left(\frac{y}{z}\right), v_P(1) \right\} \\
&= \min\{-10, -9, -8, -5, 0\} = -10,
\end{aligned}$$

and the first equality holds because the valuations of all summands are distinct. Summarizing,

$$\operatorname{div}(f_h) = P_1 + P_2 + P_3 + P_4 + P_5 + 2P_6 + \hat{D} - 10P,$$

where $\hat{D}$ is the divisor for all zeros (of any multiplicity) over all extension fields. Note that $f(x, y)$ is a polynomial, so the homogenized rational function $f(x, y, z)$ cannot have poles outside infinity. •

In projective coordinates only those rational functions are considered for which numerator and denominator are a polynomial of the same degree. Due to this, the following precise statement is possible [Lin90]:

Lemma 5. *Let $f(x, y, z)$ be a homogeneous rational function, then*

$$\deg\left(\operatorname{div}(f)\right) = 0.$$

Example 12. To illustrate the lemma, we continue Example 11. There, we found

$$\operatorname{div}(f_h) = P_1 + P_2 + P_3 + P_4 + P_5 + 2P_6 + \hat{D} - 10P,$$

and with $\deg(\operatorname{div}(f_h)) = 0$ it follows that $\deg(\hat{D}) = 3$. This means that $f(x, y, z)$, and consequently also its affine counterpart, has three zeros (counted with multiplicities) over some extension field. •

3.4 Riemann-Roch Spaces and the Riemann-Roch Theorem

The last topic in this chapter is the introduction of Riemann-Roch spaces and the Riemann-Roch theorem. The Riemann-Roch spaces are probably the main ingredient in the definition of an AG code, and the Riemann-Roch theorem allows to estimate the dimension of the code. While the actual Riemann-Roch theorem requires a canonical divisor, which is often presented as a result of the study of differential forms, there exists a corollary of the Riemann-Roch theorem that is sufficient for the codes considered in this thesis.

Definition 16 (Riemann-Roch Space)**.** *Let G be a divisor on a curve $\mathcal{X}$. A Riemann-Roch space is the set $\mathcal{L}(G)$ of rational functions $f(x, y, z)$*

$$\mathcal{L}(G) := \{f | \operatorname{div}(f) + G \geq 0\} \cup \{0\}.$$

$\mathcal{L}(G)$ has the structure of a vector space of dimension $l(G)$.

Of course, the corresponding affine functions can be found by dehomogenizing all polynomials in $\mathcal{L}(G)$. Recall that $\operatorname{div}(f) = \sum v_P P$ with v_P positive if f has a zero at P and negative if f has a pole at P. Consequently, if $G = \sum g_P P$, then some coefficient $g_P > 0$ allows f to have a pole of order not more than g_P in P, whereas $g_P < 0$ requires f to have a zero of multiplicity at least g_P in P. If $g_P = 0$, a function $f \in \mathcal{L}(G)$ may have a zero in P, but no pole.

Example 13. To illustrate the concept of Riemann-Roch spaces, consider again the Hermitian curve over $GF(4^2)$. As stated in Example 9

$$v_P\left(\frac{x^a y^b}{z^{a+b}}\right) = -4a - 5b, \text{ where } a, b \geq 0,$$

where P is the unique point at infinity. Because $z = 1$ for all affine points the functions $\frac{x^a y^b}{z^{a+b}}$ have no poles outside P. This knowledge alone allows us to give the monomial basis for the following (affine) spaces

$$\begin{aligned}
&\mathcal{L}(3P) : \{1\},\\
&\mathcal{L}(4P) : \{1, x\},\\
&\mathcal{L}(5P) = \mathcal{L}(6P) = \mathcal{L}(7P) : \{1, x, y\},\\
&\mathcal{L}(8P) : \{1, x, y, x^2\},\\
&\quad \dots\\
&\mathcal{L}(15P) : \{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3\}.
\end{aligned}$$

For larger spaces the monomial basis may no longer be unique: the space $\mathcal{L}(20P)$ can be described both by the basis

$$\{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, xy^3, y^4\}$$

and by

$$\{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, xy^3, x^5\}.$$

This is due to the fact that on the Hermitian curve $\mathcal{X}$ the monomials $\{x^5, y^4, y\}$ form a linearly dependent set, the dependence is given by the defining equation. Therefore, both bases generate the same set of polynomials and including either x^5 or y^4 in the basis does not change the Riemann-Roch space.

As given in Appendix A, for the point $P_0 = (0 : 0 : 1)$ one has

$$v_{P_0}\left(\frac{x^a y^b}{z^{a+b}}\right) = a + 5b, \text{ where } a, b \geq 0.$$

So for example $\mathcal{L}(5P)$ has basis $\{1, x, y\}$ but $\mathcal{L}(5P - P_0)$ has a smaller basis $\{x, y\}$ because a constant function cannot be zero in P_0. Further $\mathcal{L}(5P - 2P_0)$ has basis $\{y\}$, because x has a zero of multiplicity 1 in P_0, but the functions in $\mathcal{L}(5P - 2P_0)$ need to have at least a double zero. •

To estimate the dimension of a Riemann-Roch space, the following relations can be used:

Lemma 6. *Let G be a divisor on a curve $\mathcal{X}$. Then*

$$l(G) = 0 \ , \mathcal{L}(G) = \{0\} \text{ if } \deg(G) < 0, \tag{3.10}$$
$$l(G) \leq 1 + \deg(G) \text{ if } \deg(G) \geq 0. \tag{3.11}$$

The last relation is the corollary of the Riemann-Roch theorem that gives the exact dimension under a stronger constraint on the degree of G:

$$l(G) = \deg(G) - g + 1 \text{ if } \deg(G) \geq 2g - 1. \tag{3.12}$$

Example 14. We continue Example 13. Recall that for the given curve the genus was $g = 6$. Counting shows that e.g.

$$\begin{aligned}
l(3P) = 1 \leq 1 + \deg(3P) = 4 \quad &\checkmark\\
l(4P) = 2 \leq 1 + \deg(4P) = 5 \quad &\checkmark\\
l(8P) = 4 \leq 1 + \deg(8P) = 9 \quad &\checkmark\\
l(15P) = 10 = \deg(15P) - g + 1 \quad &\checkmark.
\end{aligned}$$

The last equation is due to (3.12), whereas the others are due to (3.11). In this special case, where $G = mP$, the genus g has another meaning: there are exactly g positive integers m for which $l(mP) = l((m-1)P)$. For the Hermitian curve over $GF(4^2)$, these are the integers

$$\{1, 2, 3, 6, 7, 11\}.$$

To illustrate the meaning of (3.10), consider the space $\mathcal{L}(5P - 7P_0) = \{0\}$. In the same way as shown at the end of Example 13, one finds that $v_{P_0}(\varphi) < 7$ for all basis elements $\varphi \in \mathcal{L}(5P)$, so none of them lies in $\mathcal{L}(5P - 7P_0)$ and the statement follows. •

With the help of the Riemann-Roch spaces, it is now possible to define the *ring of polynomials* on a Hermitian curve. Later, in the description of the division algorithm, all polynomials that are used belong to this ring.

Definition 17. *The ring $\mathcal{R}$ of polynomials on a Hermitian curve is*

$$\mathcal{R} = \bigcup_{m=0}^{\infty} \mathcal{L}(mP).$$

From now on, unless stated otherwise, all operations are performed in this ring of polynomials. As mentioned before, the basis for $\mathcal{L}(mP)$ is not unique for large m. Throughout the thesis, we use the set

$$\Phi = \{\varphi_{a,b} = x^a y^b | 0 \leq a \leq q, 0 \leq b\} \tag{3.13}$$

as basis for $\mathcal{R}$. This means that if the result of any calculation has $\deg_x(f) > q$, then this degree needs to be reduced with the help of the defining equation of the curve.

Chapter 4

Algebraic-Geometric Codes

With the algebraic basics introduced in the previous section, it is now possible to define algebraic-geometric (AG) codes. While the term "AG code" refers to a wide variety of codes, we use it throughout this thesis to refer to codes that can be described as evaluation codes with the help of two divisors: one divisor fixes the evaluation points of the code, the other defines a Riemann-Roch space, and the code is constructed by evaluating functions from this Riemann-Roch space at the points given by the first divisor. Although not necessary, we use only codes where the evaluation points are (a subset of) the affine points of some curve $\mathcal{X}$ because this choice induces nice structural properties to the code.

In this chapter, we first give a general definition of AG codes. Where possible, their basic parameters (i.e., the length, dimension and minimum distance) or bounds on these parameters are given. Sometimes, the class of codes we define here are referred to as *geometric generalized RS codes* as opposed to the *geometric Goppa codes* (cf. [Lin90]). The definition of the latter is based on differential forms (that have not been introduced), but it does not yield different codes: a geometric generalized RS code can always be represented as a geometric Goppa code, and it is known how the respective design parameters have to be chosen to obtain the same codes, cf. e.g. [HLP98, Thm. 2.72].

To illustrate the general definition, two special subclasses of AG codes are described in more detail. The first class are the already known Reed-Solomon codes, this time defined as AG codes to strengthen the motivation of generalizing their decoding algorithms to AG codes. The second class of codes is the class of Hermitian codes. As the name implies, these codes are defined with the help of Hermitian curves. For these two classes of codes, an interesting property is that the dual of a code belongs to the same class of codes. For RS codes this fact had already been introduced in Section 2.2, and for Hermitian codes the exact relation is given after their definition.

In the last section, we give an alternative definition for Hermitian codes that uses only those algebraic basics that were already used in the definition of RS codes. The purpose of this alternative definition is partly to introduce some notations for Chapters 5 and 6, but mainly to demonstrate that by choosing any special kind of AG code, it is possible to describe and use this very kind of code without the need to study algebraic geometry beforehand. The description of

the algorithms in the upcoming chapters is such that it can be understood from the second definition alone, but some of the proofs rely on algebraic geometry.

4.1 Algebraic-Geometric Codes and Their Parameters

The following definition of AG codes is a generalization of Definition 3, as the example of RS codes given in the next section illustrates.

Definition 18 (Algebraic-Geometric Code)**.** *An AG code $\mathcal{AG}(D, G)$ is described by a divisor $D = P_1 + P_2 + \cdots + P_n$ and an arbitrary divisor G with* $\operatorname{supp}(D) \cap \operatorname{supp}(G) = \emptyset$*:*

$$\mathcal{AG}(D, G) = \{(f(P_1), f(P_2), \ldots, f(P_n)) \,|\, f \in \mathcal{L}(G)\},$$

i.e., the code is constructed by evaluating all functions $f \in \mathcal{L}(G)$ at the points given in D.

Note that the divisor D does not give any information about zeros or poles of any rational function actually occurring in the encoding process, instead of this divisor a list of points could be given. The divisor G is an arbitrary divisor on a given curve $\mathcal{X}$, defining the Riemann-Roch space of functions that are to be evaluated. An implicit restriction on the divisor G is $\deg(G) \geq 0$, otherwise the resulting code becomes trivial. If G contains only one point, i.e.,

$$G = mP \text{ with } m > 0,$$

then the code is called a *one-point code*. In this thesis, we treat only one-point codes (with one exception) as these allow to construct the codes with the maximum code length for a given curve. Further, as can be seen in the last section of this chapter, such one-point codes usually exhibit a very nice structure that allows a description of the specific code with only very few basic algebraic objects and methods.

Though it is not necessary, two common restrictions on the divisor G are

$$2g - 2 < \deg(G) < n = \deg(D),$$

where g is the genus of the underlying curve. The second restriction is used to obtain an injective mapping (without this restriction, no statement about the dimension is possible), whereas the first restriction allows a proficient application of (3.12):

Theorem 7. *Let $2g-2 < \deg(G) < n = \deg(D)$. Then the AG code $\mathcal{AG}(D, G)$ has dimension $k = \ell(G) = \deg(G) - g + 1$, and its minimum distance is bounded by $d \geq d^* = n - \deg(G)$, where d^* is called the designed minimum distance.*

The proofs of these properties are given in [Lin90]: the dimension k is a straightforward consequence of the Riemann-Roch theorem under the condition $\deg(G) < n$, and the minimum distance of the code follows from the fact that $\deg(G)$ is the maximum number of zeros that a polynomial in the Riemann-Roch space $\mathcal{L}(G)$ can have outside $\operatorname{supp}(G)$. Unfortunately, it is usually the designed

minimum distance that limits the decoding capabilities of algebraic decoders, whereas a BMD decoder should be able to correct all errors up to half the *actual* minimum distance. But this is not a big issue: for Hermitian codes, the exact minimum distance is known and furthermore $d = d^*$ for typical code rates, i.e., for code rates that are neither very small nor very large. More details on the latter part can be found in Lemma 10. For other kinds of AG codes the Singleton bound states that the difference between the designed minimum distance and the actual minimum distance is upper bounded by g if $\deg(G) > 2g-2$: namely

$$d \leq n - k + 1 = n - (\deg(G) - g + 1) + 1 = n - \deg(G) + g.$$

In the next two sections, we introduce RS codes and Hermitian codes as special subclasses of AG codes. If these codes have some useful properties that are specific to each subclass, those properties are given, too.

4.2 Special Case: Reed-Solomon Codes

The reason for choosing RS codes in all examples in Chapter 2 is that they can be seen as a special subclass of AG codes:

Theorem 8. *An $\mathcal{RS}(n,k)$ code can be described as an $\mathcal{AG}(D,G)$ code over $GF(q)$ on the curve with defining equation $z = 0$, with the divisors $G = (k-1)P$, $P = (1:0:0)$ and $D = \sum_{i=0}^{n-1}(\alpha^i : 1 : 0)$ with $n = q-1$, and α is a primitive element of $GF(q)$.*

Of course, the point $P_0 = (0:1:0)$ could also be included in the divisor D. But the resulting code would then be one that is commonly called an *extended RS code*, so to be consistent with Chapter 2 it is left out in the given definition.

Proof. With the defining polynomial $h(x,y,z) = z$ the curve on which RS codes are defined is the projective line that lies at in infinity on the projective plane over $GF(q)$. It consists of the "double-projective" point $P = (1:0:0)$, the point $P_0 = (0:1:0)$ and the points[7] $P_i = (\alpha^i : 1 : 0)$ with $0 \leq i \leq n-1$. Implicitly, the lemma also states that RS codes are one-point codes based on the Riemann-Roch spaces $\mathcal{L}((k-1)P)$.

To find the necessary Riemann-Roch spaces, it is first necessary to find the valuations of the basic monomials in P. The tangent to this point is $\tau_P = z$, so $t_P = \frac{y}{x}$ is a possible local parameter. A basis for $\mathcal{L}((k-1)P)$ is hence given by the set

$$\left\{1, \frac{x}{y}, \frac{x^2}{y^2}, \ldots, \frac{x^{k-1}}{y^{k-1}}\right\}.$$

The functions in $\mathcal{L}((k-1)P)$ shall be evaluated in all points $P_i = (\alpha^i : 1 : 0)$, $i = 0, \ldots, n-1$. In all these points $y = 1$, so it suffices to write the functions in $\mathcal{L}((k-1)P)$ as polynomials, and the basis of the space becomes

$$\{1, x, x^2, \ldots, x^{k-1}\}.$$

[7] Note that while the points P_i are points at infinity when taking the projective plane into account, they are affine points if one only considers the projective line obtained by dropping the (zero-valued) z-coordinate from each point.

The polynomials are then evaluated at all α^i, $i = 0, \ldots, n-1$. Comparing this result to Definition 3 it becomes clear that an $RS(n, k)$ code has been constructed. □

The dimension of the RS code can also be derived with the tools presented in the previous section: the genus of the projective line is $g = 0$ according to the Plücker formula, so the Riemann-Roch theorem allows to calculate the code dimension if $m = k - 1 > -2$. As mentioned before, $m \geq 0$ is necessary to obtain a nontrivial code, so it is possible to derive the code dimension k_c from the definition $m = k - 1$ for all admissible m:

$$k_c = m - 0 + 1 = k.$$

Note that the notation k_c was chosen to allow a clear distinction between the code dimension k_c and the design parameter k (that happen to be the same in this case).

It is also possible to obtain the more general form of RS codes introduced in Definition 4: for $k_0 \neq 0$ choose the divisor $G = (k - 1 + k_0)P - k_0 P_0$. A basis for $\mathcal{L}(G)$ then is

$$\left\{ \frac{x^{k_0}}{y^{k_0}}, \frac{x^{k_0+1}}{y^{k_0+1}}, \ldots, \frac{x^{k_0+k-1}}{y^{k_0+k-1}} \right\},$$

and the rest follows analogously. Note that for this new form the requirement $\mathrm{supp}(D) \cap \mathrm{supp}(G) = \emptyset$ now prohibits to use P_0 as an evaluation point.

4.3 Special Case: Hermitian Codes

For Hermitian codes, the situation is slightly more complex than for RS codes. It is not surprising that Hermitian codes are defined with the help of a Hermitian curve $\mathcal{X}$. Recall that for a Hermitian curve

$$h(x, y, z) = x^{q+1} - y^q z - y z^q$$

is the defining polynomial over the finite field $GF(q^2)$. For the points on $\mathcal{X}$ we use the notation introduced in Section 3.2. As further derived there, this curve has q^3 affine points and a single point at infinity. Using the point at infinity in the divisor G one obtains a one-point code that is evaluated in affine points only. This choice allows to give a definition in the next section that does not require the use of projective coordinates.

Definition 19 (Hermitian Codes - with Algebraic Geometry)**.** *Let $\mathcal{X}$ be a Hermitian curve over $GF(q^2)$ and set the divisor $G = mP$, with $P = (0, 1, 0)$ the unique point at infinity of $\mathcal{X}$. Further, define the divisor $D = \sum_{i=1}^{q^3} P_i$, i.e., the formal sum of all affine points on $\mathcal{X}$. Then the Hermitian code $H(m)$ is given as the AG code $\mathcal{AG}(D, mP)$.*

Of course, the resulting code would still be a Hermitian code if one or more points were left out in the definition of D. However, the reason for fixing the length to $n = q^3$ is not only to make the codelength as long as possible, but some of the properties presented in this section only hold if the evaluation is performed in all points.

To find the generator matrix of a Hermitian code recall from Example 13 that a basis of the Riemann-Roch space $\mathcal{L}(mP)$ is given by

$$\Phi_m = \{x^a y^b | 0 \leq a \leq q, 0 \leq b, qa + (q+1)b \leq m\},$$

so the code is constructed by evaluating bivariate polynomials of restricted degree. This is the basis of the alternate description of the codes in the next section. From the description as polynomial evaluation the generator matrix follows in the same way as for RS codes:

$$\underline{\mathbf{G}} = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ x_1 & x_2 & x_3 & x_4 & \dots & x_n \\ y_1 & y_2 & y_3 & y_4 & \dots & y_n \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & \dots & x_n^2 \\ x_1 y_1 & x_2 y_2 & x_3 y_3 & x_4 y_4 & \dots & x_n y_n \\ y_1^2 & y_2^2 & y_3^2 & y_4^2 & \dots & y_n^2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix}. \tag{4.1}$$

Again, the results from Section 4.1 now allow to estimate the code parameters: the Hermitian curve has genus $g = \frac{1}{2}(q^2 - q)$, so

$$\begin{aligned} &k = m - g + 1 \quad \text{if} \quad m \geq 2g - 2 = q^2 - q - 2 \text{ and} \\ &d \geq d^* = n - m. \end{aligned} \tag{4.2}$$

But beyond all these parameters, the most useful property in the decoding process is the fact that the dual of the Hermitian code $H(m)$ is again a Hermitian code.

Theorem 9 (Dual Hermitian Code [JH04])**.** *Let $H(m)$ be a Hermitian code over $GF(q^2)$ with length $n = q^3$ according to the above definition. Then the dual to this code is the Hermitian code $H(m^\perp)$ where*

$$m^\perp = n + q^2 - q - 2 - m. \tag{4.3}$$

Proof. Because it is a simple task to find the generator matrix of a Hermitian code, (4.3) makes is an equivalently simple task to find the parity check matrix $\underline{\mathbf{H}}$ of a Hermitian code $H(m)$. The matrix product $\underline{\mathbf{H}} \cdot \underline{\mathbf{G}}^T$ can be used to prove the duality in a similar way as for RS codes in Section 2.2, but the relations get a little more involved due to the bivariate monomials that appear. More details on this proof are found in [JH04]. □

Another equivalence to RS codes is that if the matrix were extended with more monomials, these additional relations would allow to reconstruct the information word in the error-free case. Therefore, the syndrome values obtained from the check matrix $\underline{\mathbf{H}}$ are mapped to a syndrome polynomial $S(x, y)$ in a way similar to the mapping for RS codes. Details of this mapping are given in Section 5.1.

Using the relation between m and $m^\perp$ allows to improve the general estimate for the minimum distance: a closed formula for the *actual* minimum distance is given e.g. in [Duu08] and [HLP98]. Because no exact bounds on the allowed values of m are given in [Duu08], we use the table from [HLP98, Section 5.3].

Lemma 10. *Consider a Hermitian code $H(m)$ with design parameter $2g-1 < m < n-2g+1$. Then $d = d^* = n - m$.*

Proof. The Hermitian code $H(m)$ is denoted as $E_{l^\perp}$ in [HLP98]. In the range specified for the design parameter m the relation between the parameters is $l^\perp = m - g + 1$. The code $E_{l^\perp}$ is equivalent to some code C_l for which the minimum distance is known: specifically the minimum distance of C_l is $d_C = l + g - 1$ if $3g - 2 < l < n - g$. Since the codes are equivalent this means that

$$d_E = l^\perp + g - 1 \text{ if } 3g - 2 < l^\perp < n - g,$$

and substituting the relation between $l^\perp$ and m yields

$$d = n - m$$

for the design parameter m used here and the bounds on $l^\perp$ are equal to the bounds given for m before. □

The lemma shows that for codes with rates that are neither too large nor too small, the bound on the minimum distance is fulfilled with equality. All codes used in the examples and simulations in this thesis were in this range. Further, we always assume that the minimum distance is $d = n - m$ in proofs and derivations, implicitly limiting the allowed design parameters to the range given in Lemma 10.

4.4 Defining Hermitian Codes without Algebraic Geometry

Of course the heading of this section is slightly paradox: to define a code on a curve it is always necessary to have a curve first and this is an algebraic-geometric object. But apart from that, the knowledge necessary to understand RS codes also suffices for Hermitian codes: there is no need to study the entire basics of algebraic geometry if one wants to apply Hermitian codes (or any other specific kind of AG code) only, but not general AG codes. In this section, based on the results of the previous section, we give such a definition. This definition, and many of the notations used, are adopted from [JH04]. However, just as for RS codes, using the special definition might keep one from seeing the bigger framework in which Hermitian codes are settled and from the possibility to extend the found results to other codes. To make the connection to the results obtained before with the help of algebraic geometry, we give the relations in brackets.

To describe Hermitian codes in a way similar to RS codes, it is first necessary to fix something like the degree of a bivariate polynomial on a curve. If the underlying curve $\mathcal{X}$ is the Hermitian curve over $GF(q^2)$ with defining polynomial $h(x, y) = x^{q+1} - y^q - y$, the following definition is used:

Definition 20 (Degree of a Bivariate Polynomial)**.** *For a monomial $x^a y^b$, the bivariate degree is given by the function*

$$\rho(x^a y^b) = qa + (q+1)b.$$

The degree of a polynomial of the form $f(x,y) = \sum f_{a,b} x^a y^b$ *is given by*

$$\rho(f) = \max\{\rho(x^a y^b) | f_{a,b} \neq 0\}.$$

(This definition of the bivariate degree is equal to the negative of the pole order of $f(x,y)$ at the point at infinity as found in Example 9.) Note that this definition implies a difference between the degree of a polynomial on a curve and the degree of a polynomial over the entire plane used in Chapter 3 (e.g. in the Plücker formula). From now on, only polynomials on a Hermitian curve are used.

As before, all polynomial calculations have to be performed modulo $h(x,y)$. Due to this, the set

$$\Phi = \{\varphi_{a,b} = x^a y^b | 0 \leq a \leq q, 0 \leq b\}$$

is a basis for all polynomials on the curve. For a shorter notation in the rest of the thesis, also define the set

$$\Phi_m = \{\varphi_{a,b} \in \Phi | \rho(\varphi_{a,b}) \leq m\}.$$

(Φ_m is the basis of the Riemann-Roch space $\mathcal{L}(mP)$, and Φ was already given in (3.13) as the basis for the ring of polynomials on the Hermitian curve.) For the polynomials in these sets, two indexing systems are used, each of them having its advantage: the system with double indices, that was already used in the definitions of Φ and Φ_m, allows to directly obtain the exponents of the monomial. In the description of the algorithm, another indexing system with single indices is used. This other system refers to the natural ordering of the monomials implied by the degrees, i.e., $\varphi_0 = 1$, $\varphi_1 = x$, $\varphi_2 = y$, $\varphi_3 = x^2$, $\varphi_4 = xy$ and so on. Because the degrees of all monomials in Φ are unique, this ordering is also unique.

With the help of these definitions, it is now possible to repeat Definition 19.

Definition 21 (Hermitian Codes - without Algebraic Geometry)**.** *The codewords of a Hermitian code* $H(m)$ *over* $GF(q^2)$ *with design parameter* m *are obtained by evaluating polynomials over* $GF(q^2)$ *of degree* $\rho(f) \leq m$*, i.e.,*

$$f(x,y) = \sum_{\varphi_i \in \Phi_m} a_i \varphi_i(x,y), \tag{4.4}$$

with $a_i \in GF(q^2)$*, at all points on the affine Hermitian curve* $\mathcal{X}$ *over* $GF(q^2)$*.*

Of course, the code parameters are still the same as in (4.2), and it is relatively simple to verify these results with basic algebraic methods: the dimension of the code follows from counting $|\Phi_m|$ and is related to the Frobenius coin problem[8]. The designed distance d^* can be derived with the arguments on the number of zeros of a polynomial given in Section 4.1: a polynomial $f(x,y)$ with $\rho(f) = m$ has at most m zeroes. Unfortunately, a bivariate polynomial on a curve cannot be factored in the same way as a univariate polynomial, where each root is equivalent to one factor of the polynomial, so no descriptive derivation of this fact exists. Nevertheless, the bound on the number of zeros of a polynomial

[8]This problem is also known as coin problem or, in German, Briefmarkenproblem (stamp problem).

shows that each codeword has weight $w_H(\underline{\mathbf{c}}) \geq n - m$, and the bound on the minimal distance given in Theorem 7 follows.

Because the generator matrix obtained for codes defined in this section is exactly the same as (4.1) in the previous section, it becomes clear why the proof from [JH04] could be used to prove the duality

$$H(m)^{\perp} = H(m^{\perp})$$

already in the previous section. No second proof is given, because no other methods than in the previous section would be used.

Chapter 5

A Division Decoding Algorithm for Hermitian Codes

In this section, we describe a new decoding algorithm for Hermitian codes. The main idea behind the design of this algorithm was to obtain an equivalent to the extended Euclidean algorithm used for the decoding of RS codes (see Section 2.3). In 1988 and 1992, Porter [Por88] and Shen [She92] already published algorithms that are usually cited as equivalent to the EEA. However, their algorithms both rely on the construction of a so-called subresultant sequence and the main operations in their algorithms are matrix manipulations. While a subresultant sequence provenly yields the same results as the EEA for univariate polynomials, its description is much different. In contrast to that, the algorithm presented here uses repeated divisions of polynomials and hence its description is much closer to the EEA.

Unfortunately, the use of bivariate polynomials and the division algorithm for bivariate polynomials make the algorithm more complex than the decoding algorithm for RS codes. Especially, it is no longer sufficient to store only two previously determined remainder polynomials as was the case for the EEA. This increased number of remainder polynomials is also the reason that decoding is no longer possible with quadratic complexity.

This chapter is organized as follows: in the first section the key equation for Hermitian codes is introduced, and its solution is characterized. We give only a sketch of the proof for uniqueness of the solution, but details are given in [BK] and alternative versions of both the key equation and the proof can be found in many places in literature (e.g. [PSP92], [OBA08]). After that we state the division problem for bivariate polynomials, because repeated divisions are the core part in our decoding algorithm. In Section 5.3 we describe the basic algorithm and illustrate its functionality with an example. Additionally, a short proof of correctness of the algorithm is given. The basic algorithm is only capable of locating *all* error patterns with weight is not larger than half the minimum distance with an extension. This extension is given in Section 5.5 and also illustrated with an example. To conclude the section, we give the complexity of the algorithm and its extension in Section 5.6.

Most of the results of this chapter, especially the key equation and its proof,

had been found in joint work with Prof. Irene Bouw from the Institute of Pure Mathematics at the University of Ulm.

5.1 Syndromes and the Key Equation

There are a lot of publications, each presenting the key equation for Hermitian codes in a slightly different form, so it seems needless to present it again in yet another form. But most of the other descriptions use a lot of algebraic geometry, making the conditions hard to understand. In this thesis, we give the key equation in a polynomial form, making its representation very similar to the key equation for RS codes given in (2.6). Besides, the key equation is also given in a matrix form that is very useful in several proofs.

Because the key equation for Hermitian codes requires a syndrome polynomial, too, this is given first.

Syndrome Elements and Syndrome Polynomial

To define the syndrome polynomial, it is first necessary to define the syndrome elements

$$s_{a,b} = \sum_{j=1}^{n} e_j \varphi_{a,b}(P_j) = s_i \quad \text{if } \varphi_i = \varphi_{a,b}. \tag{5.1}$$

While this definition is given for all $\varphi \in \Phi$, it is not always possible to calculate the syndromes from the received word $\underline{\mathbf{r}}$: if $\rho(\varphi_i) \leq m^{\perp}$ one has

$$\sum_{j=1}^{n} c_j \varphi_{a,b}(P_j) = 0$$

by the definition of the check matrix and the dual code, for larger $\rho(\varphi_i)$ the sum can have arbitrary values. Because an additive error was assumed and the calculation of the syndromes is a linear operation, the respective syndromes can be calculated from the received word:

$$s_{a,b} = \sum_{j=1}^{n} r_j \varphi_{a,b}(P_j) \quad \text{if} \quad \rho(\varphi_{a,b}) \leq m^{\perp}.$$

These syndromes are therefore called the *known* syndromes, whereas the syndromes $s_{a,b}$ where $\rho(\varphi_{a,b}) > m^{\perp}$ are referred to as *unknown* syndromes. Of course, only the known syndromes can be used in the decoding process. Just as for RS codes, these syndromes are next mapped to coefficients of a syndrome polynomial.

Definition 22 (Syndrome Polynomial for Hermitian Codes)**.** *The syndrome polynomial $S(x, y)$ for Hermitian codes is defined as*

$$S(x, y) = \sum_{\rho(x^a y^b) \leq m^{\perp}} s_{a,b} x^{a_m - a} y^{b_m - b}, \tag{5.2}$$

where $a_m = q$ and $b_m = \max\{b : \rho(x^a y^b) \leq m^{\perp}\}$. Further, denote by

$$\rho_S = a_m q + b_m (q + 1)$$

the maximum possible degree of $S(x, y)$.

The choice $a_m = q$ is necessary because all operations are performed modulo the defining equation of the curve. Choosing a smaller a_m leads to wrong results because the modulo operation will not work as intended, a larger a_m is not allowed for polynomials on the curve. On the other hand, the choice of b_m creates a syndrome polynomial of minimal degree. It is also possible to choose b_m larger, but there is no gain in doing so at the cost of dealing with polynomials of larger degree. Note that this definition mimics the definition of the syndrome polynomial for RS codes very closely (see Section 2.2). The importance of the value ρ_S becomes clear through all the occasions in which it is used later on.

In some situations, usually when looking at single coefficients of the polynomial product $\Lambda(x, y)S(x, y)$, it is helpful to define syndrome elements $s_{a,b}$ with $a > q$ to avoid the calculation modulo the defining equation of the curve. These syndromes can be obtained from the same definition by simply using the respective monomial (even though it is not allowed as a term in a function on the curve) or by using the relation

$$s_{a,b} = s_{a-q-1,b+1} + s_{a-q-1,b+q}.$$

Such syndromes are usually called *inferred* syndromes; they are known if both $s_{a-q-1,b+1}$ and $s_{a-q-1,b+q}$ are known, otherwise they are unknown. Of course, unknown inferred syndromes cannot be used in the decoding process either.

Error Locator Polynomials

Another important notion that we use is that of the error locator polynomial. While trying to use a notion that is as close as possible to the respective notion of Section 2.3, the use of bivariate polynomials requires some slightly alternative definitions.

Definition 23 (Error Locator Polynomial)**.** *An* error locator *polynomial is a polynomial* $\Lambda(x, y) = \sum_{j=0}^{i} \lambda_j \varphi_j$, $\lambda_i \neq 0$, *that has at least* i *distinct affine zeros. Given an error word* $\underline{\mathbf{e}}$, *a* correct *error locator is an error locator polynomial* $\Lambda(x, y)$ *that has the property* $\Lambda(P_i) = 0$ *if* $e_i \neq 0$, *and an error locator is* minimal *if, for the same error positions, there is no error locator of smaller degree.*

The reason for requiring a certain number of zeros is that any error of weight i can be located with a polynomial having at most $i+1$ terms, so it is possible to search an error locator only among the polynomials of restricted degree. Note that this definition of an error locator implicitly requires the polynomials to be considered on a curve that allows a single indexing of the basis monomials based on their monomial orders. Such an ordering always exists for one-point codes due to properties of the valuation. As shown in the previous chapter, such an indexing exists for Hermitian curves. Consequently, the zeros are only counted among the points on the curve.

Example 15. Consider the Hermitian curve $\mathcal{X}$ for $q = 4$ as the underlying curve. Then the polynomial

$$f_1(x, y) = xy + \alpha^5 x^2 + y + \alpha^{13} x + \alpha^8$$

is not an error locator: its leading term is it $\varphi_4 = xy$, so $i = 4$ and by definition this would require $f_1(x, y)$ to have at least four zeros. But $f_1(x, y)$ has only

three zeros on the curve, namely in the points (α, α^6), (α^2, α^3) and (α^5, α^{11}), so it is not a locator.

To illustrate the second part of the definition, assume that an error corrupted positions $(1, \alpha)$ and $(1, \alpha^2)$. Then both

$$f_2(x, y) = x + 1 \text{ and } f_3(x, y) = x^2 + 1$$

are correct error locators. Clearly $f_3(x, y)$ cannot be a minimal error locator, and to see that $f_2(x, y)$ actually is a minimal error locator note that a polynomial can only have smaller degree if it is constant, but then it has no zeros. •

Compared to the error locators used for RS codes, the minimal error locator for Hermitian codes can have some additional zeros. It follows from basic algebraic geometry that $\rho(\Lambda) \leq t + g$ where t is the error weight: any pattern of t points can be found among the zeros of a polynomial that has t (or less) free coefficients, so $\rho(\Lambda) \leq \rho(\varphi_t) \leq t + g$. On the other hand, a polynomial of degree $t + g$ has at most $t + g$ zeros[9], and because t of these zeros must be the error positions then at most g additional zeros can be present. This property also implies that for a given error weight, the degree of the minimal error locator is not fixed, but only an upper and lower bound can be given:

$$\rho(\Lambda) - g \leq t \leq \rho(\Lambda),$$

the lower bound follows from the definition of an error locator and the fact that $\rho(\varphi_t) \leq t + g$ and the upper bound is the fundamental theorem of algebra.

The Key Equation

The key equation for Hermitian codes can be stated only for an error weight that is bounded by some value smaller than half the minimum distance. The parameter s appearing in this bound is the *Clifford defect* of a curve that can be calculated according to the following formula [BK], [PSP92]:

$$s = \begin{cases} (q-1)^2/8 + 1/2, \text{ if } q \equiv 1 (\text{mod } 2), \\ (q-2)^2/8 + 1/2, \text{ if } q \equiv 0 (\text{mod } 2). \end{cases} \tag{5.3}$$

Note that the value s used without any index denotes the Clifford defect, whereas a syndrome element always has a single or double index to relate it to the respective monomial.

Theorem 11 (The Key Equation for Hermitian Codes)**.** *Let* $\underline{\mathbf{e}}$ *be an error of weight* $t \leq \left\lfloor \frac{d-1}{2} \right\rfloor - s$*, and* $S(x, y)$ *the syndrome polynomial calculated by* (5.1) *and* (5.2)*. Then there exist a unique minimal error locator* $\Lambda(x, y)$ *and a corresponding error evaluator polynomial* $R(x, y)$ *that fulfill*

$$\Lambda(x, y) \cdot S(x, y) = R(x, y) \mod y^{b_m + 1}, \tag{5.4}$$

under the constraint

$$\rho(R) - \rho(\Lambda) \leq q a_m + (q+1) b_m - m^{\perp} - 1 =: \ell, \tag{5.5}$$

and $\rho(\Lambda)$ *is minimal among all pairs* (Λ, R) *satisfying* (5.4) *and* (5.5)*.*

[9] The number of zeros counted with multiplicities equals $t + g$ if the curve is defined over an appropriate algebraic extension of the base field.

Note that the key equation actually consists of two equations: only if a pair of polynomials fulfills both (5.4) and (5.5) this pair is called a solution of the key equation. Further, the uniqueness of the solution is a direct consequence of a corollary to the Riemann-Roch theorem.

The proof of this theorem can be split into two parts: first, we prove that the correct error locator polynomial $\Lambda(x, y)$, along with a properly chosen polynomial $R(x, y)$, always fulfills (5.4) and (5.5). Because this statement alone is used again later, we state and prove it as a separate lemma afterwards (Lemma 12). The second part is the proof that, under the given constraint on the error weight, any solution must be a correct error locator. For the solution of minimal degree (under the given bound on the error weight) uniqueness follows from the fact that for the minimal μ where the space $\mathcal{L}(\mu P - Q)$ is not trivial (i.e., it contains nonzero functions) it has dimension 1, so the solution is unique up to multiplication by a constant. Alternative versions of both the key equation and the corresponding proof can be found in several papers, e.g. [Ehr91], [PSP92], [OBA08]. A special treatment is required for the point $(0 : 0 : 1)$, so in the proof we use a slightly different notation than in the rest of the thesis denoting $P_0 = (0 : 0 : 1)$ and the other affine points on the Hermitian curve by $P_1, \ldots, P_{n-1}$.

Proof. For each point $P_i = (\alpha_i, \beta_i)$ on $\mathcal{X}$ define the function

$$u_i(x, y) = \frac{1 + \sum_{j=0}^{q-1}(y^j \beta_i^{q-1-j})}{x - \alpha_i},$$

which are combined to

$$U(x, y) := -\sum_{i \in I} e_i \beta_i^{b_m+1} u_i.$$

This rational function has single poles at the error positions but at no other points except possibly the point at infinity. The extended syndrome polynomial[10]

$$\tilde{S}(x, y) = \sum_{a \le a_m, b \le b_m} s_{a,b} x^{a_m - a} y^{b_m - b}$$

is an approximation to $U(x, y)$, specifically [BK, Lemma 2.2] states that

$$\tilde{S}(x, y) = \sum_{i \in I} e_i (y^{b_m+1} - \beta_i^{b_m+1}) u_i, \tag{5.6}$$

or in other words $\tilde{S}(x, y) = U(x, y) \mod y^{b_m+1}$. Using these definitions, we first show that for any pair (Λ, R) that is a solution to the key equation

$$R - \Lambda U \in \mathcal{L}((\mu + \ell)P + Q - (q + 1)(b_m + 1)P_0), \tag{5.7}$$

where $\mu = \rho(\Lambda)$ and $Q = \sum P_{e_i}$ is the divisor containing all the error positions. The meaning of this space is the following: the pole order at infinity (which is equivalent to the degree of a bivariate polynomial) is limited to $\mu + \ell$. The function U is defined appropriately such that this pole order is dominated by the polynomial R and the limitation becomes equivalent to (5.5). The divisor Q

[10] Compared to [BK] the notations $S(x, y)$ and $\tilde{S}(x, y)$ are interchanged.

allows the function $R - \Lambda U$ to have poles in the error positions: both R and Λ are polynomials (so they do not have poles outside infinity), but U has the error positions at poles. Theoretically the minimal solution to the key equation might not be an error locator, so the poles of U are not necessarily compensated by zeros of Λ. Finally the term $-(q+1)(b_m+1)P_0$ represents the modulo operation in the key equation, because $\tilde{S}(x,y)$ was an approximation to $U(x,y)$ up to y^{b_m}.

The definition of U implies that the poles of $R-\Lambda U$ are contained in $\{P_i\}_{i\in I}\cup \{P\}$. Moreover, in P_i with $i \in I\setminus\{0\}$, the rational function $R-\Lambda U$ has at most a simple pole. The order of the pole in P equals $-\rho(R-\Lambda U)$. Since $\rho(\Lambda)=\mu$ by definition, we conclude that

$$\rho(R-\Lambda U) \leq \rho(\Lambda) + \max\left(\rho\left(\frac{R}{\Lambda}\right), \rho(U)\right).$$

Since (Λ, R) is a solution to the key equation (5.4), it follows that $\rho(R) - \rho(\Lambda) \leq \ell$. The definition of U implies that $\rho(U) \leq (q-1)(q+1)-q = q^2-q-1 = 2g-1$, so we conclude that

$$\rho(R-\Lambda U) \leq \mu + \max(\ell, 2g-1).$$

The definition of b_m implies that

$$(q+1)b_m \leq m^\perp \leq (q+1)b_m + q,$$

therefore $2g-1 \leq \ell$. We conclude that $\rho(R-\Lambda U) \leq \mu + \ell$.

It remains to estimate $v_{P_0}(R-\Lambda U)$. (5.6) states that

$$\tilde{S} - U = (\sum_{i\in I} e_i u_i) y^{b_m+1}.$$

We conclude that $v_{P_0}(\tilde{S}-U) \geq v_{P_0}(y^{b_m+1}) = (q+1)(b_m+1)$ if $0 \notin I$. In the case that $0 \in I$, we have $u_0 = (y^{q-1}+1)/x$, and hence

$$\tilde{S} - U = e_0 \frac{y^{b_m+1}(y^{q-1}+1)}{x} + \sum_{i\in I\setminus\{0\}} e_i u_i y^{b_m+1}.$$

We conclude that $v_{P_0}(\tilde{S}-U) = (b_m+1)(q+1)-1$. This shows that (5.7) is fulfilled for either case.

Since (Λ, R) is a solution to the key equation (5.4), it follows that $T = \tilde{S}\Lambda - R$ satisfies $v_{P_0}(T) \geq (q+1)(b_m+1)$. We may write $R - \Lambda U = \Lambda(\tilde{S}-U) - T$. Therefore it follows that

$$v_{P_0}(R-\Lambda U) \geq (q+1)(b_m+1) - 1.$$

Now, the aim of the proof is to show that the space in which $R-\Lambda U$ lies is trivial, i.e., it contains only the all-zero polynomial. This implies that $U = R/\Lambda$ and, because R is a polynomial, that Λ must be zero in all error positions, hence it is a correct error locator. It is relatively simple to show this for a weaker bound than the one given in Theorem 11: if $t + \rho(\Lambda) < d^*$, then

$$\begin{aligned}\deg\left((\mu+\ell)P + Q - (q+1)(b_m+1)P_0\right) &= \mu + \ell + t - (q+1)(b_m+1) \\ &< d^* + q^2 - q - 1 - m^\perp - 1 = 0.\end{aligned}$$

According to (3.10), this means that the space is trivial. This does not yet cover the entire range for which the theorem was stated because $t + \rho(\Lambda) < d^*$ implies $t \leq \left\lfloor \frac{d^*-1-g}{2} \right\rfloor$. However, the proof for the remaining cases is more involved, so we refer to [BK] for that part.

For the syndrome polynomial $S(x, y)$ the statement follows from the observation that $S - \tilde{S}$ only contains terms $x^a y^b$ with $\rho(x^a y^b) < \rho_S - m^\perp$, and the fact that

$$\frac{\tilde{R}}{\Lambda} = \frac{R}{\Lambda} + \tilde{S} - S,$$

and $\tilde{S} - S$ is a polynomial. □

The second part of the proof was only given for the case that no error occurred in the point $P_0 = (0, 0, 1)$. Similar to the first part, a special treatment is necessary because except for $\varphi_0(x, y) = 1$, all monomials in Φ are zero in P_0. However, the result is the same, and the details are also given in [BK]. Further, the proof only shows that under the given circumstances a solution to the key equation must be an error locator, so the following lemma - which was adapted from [JLJH92] - is an essential part to the proof because it shows that at least one solution actually does exist.

Lemma 12. *Given an error word* $\underline{\mathbf{e}}$ *and the corresponding syndrome polynomial* $S(x, y)$*, a correct error locator* $\Lambda(x, y)$ *and a suitable polynomial* $R(x, y)$ *always solve the key equation.*

Proof. Define $\mu = \rho(\Lambda)$ and write the error locator polynomial as

$$\Lambda(x, y) = \sum_{\rho(\varphi_{a,b}) \leq \mu} \lambda_{a,b} \varphi_{a,b}.$$

Because no limitations were given for $R(x, y)$, it is possible to define

$$R(x, y) = \Lambda(x, y) \cdot S(x, y) \mod y^{b_m+1},$$

and $R(x, y)$ contains only monomials $\varphi_{a,b}$ with $b \leq b_m$. Clearly, the pair (Λ, R) fulfills (5.4) so it remains to show that (5.5) is also fulfilled. To see this, write each coefficient of $R(x, y)$ as

$$r_{a_t,b_t} = \sum_{aq+b(q+1) \leq \mu} \lambda_{a,b} s_{a_m-a_t+a, b_m-b_t+b}.$$

Of course, the syndromes $s_{a_m-a_t+a,b_m-b_t+b}$ for $b_m - b_t + b < 0 \Leftrightarrow b_t - b > b_m$ do not exist, so in general no statement about the value of r_{a_t,b_t} is possible if $b_t > b_m$, but this is just the part of the product that is eliminated by the modulo operation. On the other hand, the syndromes $s_{a_m-a_t+a,b_m-b_t+b}$ for $(a_m - a_t + a)q + (b_m - b_t + b)(q + 1) > m^\perp$ cannot be calculated from the received word, so for these values no statement about r_{a_t,b_t} is possible either. However, by reformulating the condition, one finds that these are exactly those values that may be nonzero according to (5.5). For the remaining values, it is possible to calculate the syndromes from the received word and it is possible to

rewrite the coefficients of $R(x, y)$ to

$$\begin{aligned} r_{a_t,b_t} &= \sum_{aq+b(q+1)\leq\mu} \lambda_{a,b} \sum_{i=1}^{n} e_i \varphi_{a_m-a_t+a,b_m-b_t+b}(P_i) \\ &= \sum_{i=1}^{n} \sum_{aq+b(q+1)\leq\mu} \lambda_{a,b} e_i \varphi_{a_m-a_t,b_m-b_t}(P_i) \varphi_{a,b}(P_i) \\ &= \sum_{i=1}^{n} e_i \varphi_{a_m-a_t,b_m-b_t}(P_i) \sum_{aq+b(q+1)\leq\mu} \lambda_{a,b} \varphi_{a,b}(P_i) \\ &= \sum_{i=1}^{n} e_i \varphi_{a_m-a_t,b_m-b_t}(P_i) \Lambda(P_i). \end{aligned}$$

By definition, $\Lambda(P_i) = 0$ if $e_i \neq 0$, hence the latter sum is always zero. Combining the results for all three ranges, it turns out that the pair (Λ, R) fulfills (5.4) and (5.5). □

The proof of this lemma also implies a descriptive interpretation of the value ℓ: assume that all syndrome elements that can be calculated from the received word are nonzero, then ℓ is the largest integer such that all terms in $S(x, y)$ have larger degree (it is not necessary that $\ell = \rho(\varphi_i)$ for some i). Conversely, the stopping criterion means that all terms of the polynomial product where all $\lambda_{a,b}$ are multiplied to known syndromes have to be zero.

This latter interpretation is illustrated in Figure 5.1 for the arbitrarily chosen values $q = 4$, $m^{\perp} = 25$ and $t = 7$. In the left part of the figure, the shaded region indicates the monomials in $S(x, y)$ to which syndrome elements are mapped. With $LM(\Lambda) = \varphi_7 = x^2y$ - this is the most common result for $t = 7$ - the frames show which syndrome elements are involved in the calculation of the element r_{a_t,b_t} if the upper right corner of a frame lies over the position $x^{a_t}y^{b_t}$. For example, the calculation of $r_{3,5}$ includes the syndrome elements

$$s_{3,5}, s_{2,5}, s_{3,4}, s_{1,5}, s_{2,4}, s_{1,5}, s_{0,5}, s_{1,4},$$

and $r_{3,5} = 0$ if $\Lambda(x, y)$ is an error locator. The other two elements $r_{4,7}$ and $r_{3,2}$ may be nonzero as their computation involves unknown syndromes. In the right part, the illustration is extended to include coefficients like $r_{1,5}$: this coefficient also has to be zero, although the frame is not overlapping with the coefficients of the syndrome polynomial. But another effect has to be taken into account here, namely the calculation modulo the defining equation of the curve. This is best illustrated by drawing the known inferred syndromes as "virtual" coefficients at negative powers of x. In this case, the three inferred syndromes are known and marked with asterisks. Now any frame overlapping the shaded area and fields with asterisks has to yield a zero result.

The Matrix Form of the Key Equation

The key equation can be mapped to a number of linear equations in the coefficients of $\Lambda(x, y)$, where the number of equations varies with $\rho(\Lambda)$. Writing these linear equations in matrix form results in a structured matrix $\underline{\mathbf{S}}$ that is used in several places later on.

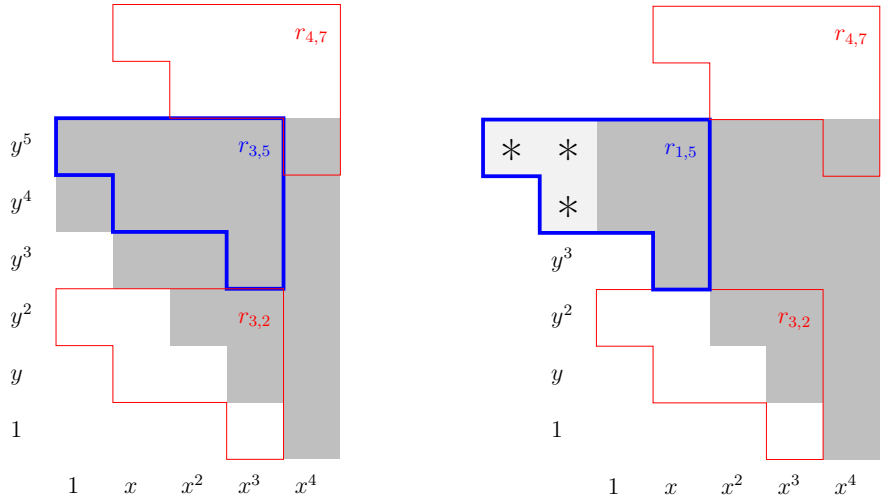


Figure 5.1: Illustration of (5.5) - without and with inferred syndromes

Consider the polynomial product $\Pi(x,y) = S(x,y)\cdot\Lambda(x,y) \mod y^{b_m+1}$. The monomial with largest degree that is not discarded by the modulo operation is $x^q y^{b_m}$. Write

$$\Lambda(x,y) = \sum_{\rho(\varphi_{a,b})\le\mu} \lambda_{a,b}\varphi_{a,b}(x,y), \quad LM(\Lambda) = \varphi_{a_l,b_l},$$

then the coefficient π_{q,b_m} is given as

$$\pi_{q,b_m} = s_{0,0}\lambda_{0,0} + s_{1,0}\lambda_{1,0} + \cdots + s_{a_l,b_l}\lambda_{a_l,b_l}.$$

Equivalently

$$\pi_{q-1,b_m} = s_{1,0}\lambda_{0,0} + s_{2,0}\lambda_{1,0} + \cdots + s_{a_l+1,b_l}\lambda_{a_l,b_l}$$

and so on. In these equations one may also need the inferred syndromes. (5.5) now shows that if $aq + b(q+1) > \ell + \rho(\Lambda)$ then $\pi_{a,b}$ must be zero, and so solving the key equation is equivalent to solving the homogeneous linear system of equations

$$\underline{\mathbf{S}} \cdot \begin{pmatrix} \lambda_{0,0} \\ \lambda_{1,0} \\ \lambda_{0,1} \\ \vdots \\ \lambda_{a_l,b_l} \end{pmatrix} = \underline{0}$$

with

$$\underline{\mathbf{S}} = \begin{pmatrix} s_{0,0} & s_{1,0} & s_{0,1} & \cdots & s_{a_l,b_l} \\ s_{1,0} & s_{2,0} & s_{1,1} & \cdots & s_{a_l+1,b_l} \\ s_{0,1} & s_{1,1} & s_{0,2} & \cdots & s_{a_l,b_l+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{a^*,b^*} & \cdots & \cdots & \cdots & s_{a_{m^\perp},b_{m^\perp}} \end{pmatrix}, \tag{5.8}$$

where $a_{m^\perp}q+b_{m^\perp}(q+1) = m^\perp$ and $a^* = a_{m^\perp} - a_l$, $b^* = b_{m^\perp} - b_l$. All syndrome elements in this matrix are known, so the number of rows of $\underline{\mathbf{S}}$ is reciprocal to $\rho(\Lambda)$ or the number of columns of $\underline{\mathbf{S}}$: more specifically, counting shows that the number of rows equals

$$d - 1 - \rho(\Lambda) + g$$

if $\rho(\Lambda) \geq 2g - 1$, or slightly larger below that bound. This means that for small $\rho(\Lambda)$ the system of equations is overdefined, so there may not exist a solution and the error cannot be located with a polynomial of the given degree. But as $\rho(\Lambda)$ increases the system eventually becomes underdefined and the solution can no longer be unique. The latter is always the case if $t > \left\lfloor \frac{d-1}{2} \right\rfloor$, and Chapter 6 shows under which circumstances an error of this weight can be corrected with a high probability. But it may also happen if $\left\lfloor \frac{d-1}{2} \right\rfloor - s < t \leq \left\lfloor \frac{d-1}{2} \right\rfloor$, and Section 5.5 illustrates how to detect and handle the latter cases.

5.2 Division of Bivariate Polynomials

Divisions of univariate polynomials are the main computations when decoding RS codes with the Euclidean algorithm (see Section 2.3), and so it is not surprising that bivariate divisions are the core part of the new algorithm. Because there are some essential differences to the univariate case we present the division procedure separately before applying it in the decoding algorithm.

The division of bivariate polynomials can be performed in a way that is very similar to the long division of univariate polynomials: the divisor is multiplied with a properly chosen monomial and a constant such that its leading term becomes the same as the leading term of the dividend, this part is often called *alignment*. Then the aligned divisor is subtracted from the dividend and the procedure is repeated with the difference, also referred to as intermediate dividend. If there is no monomial such that the leading terms can be aligned, then the leading term of the intermediate dividend is moved to the remainder $\epsilon(x, y)$. These two operations are performed until the intermediate dividend becomes zero. The two main differences of this bivariate division to the division of univariate polynomials are that the division does not stop as soon as the leading term of the dividend is not a multiple of the leading term of the divisor, and that the remainder may have larger degree than the divisor. These two properties are due to the fact that not all bivariate monomials are multiples of each other. To illustrate the bivariate division and the two mentioned characteristics consider the following example.

Example 16. Let the dividend $\theta(x, y) = x^2+y+1$ and the divisor $\xi(x, y) = x+1$ be two bivariate polynomials over $GF(2)$. Then the division works as follows:

1. Initialize the quotient $\gamma(x, y) = 0$ and remainder $\epsilon(x, y) = 0$.

2. Align ξ and θ by multiplying ξ with x.

3. $\theta_1(x, y) = \theta(x, y) - x\xi(x, y) = y + x + 1$ and $\gamma(x, y) = x$.

4. Because there is no monomial φ_i s.t. $\varphi_i \cdot x = y$, move y to the remainder, i.e., $\epsilon(x, y) = y, \theta_2(x, y) = x + 1$.

5. θ_2 and ξ are already aligned, so $\gamma(x,y) = x+1, \theta_3(x,y) = \theta_2 - \xi = 0$, and the algorithm ends.

With the determined quotient and remainder, it is now possible to write that

$$\theta(x,y) = x^2 + y + 1 = \xi(x,y) \cdot \gamma(x,y) + \epsilon(x,y) = (x+1)^2 + y. \qquad \bullet$$

In this example, the result of the division is independent of how the degree of a bivariate polynomial is defined. For some degree definitions, e.g. the one used for polynomials on Hermitian curves, one would actually find $\rho(\epsilon) > \rho(\xi)$, for others the opposite relation might be true.

Later, in the actual decoding algorithm, a polynomial shall be divided by *several* polynomials. In such a case, there are several quotients, but also just a single remainder polynomial. Because this division by several polynomials is an essential part, we state the division problem more formally.

Definition 24 (The Bivariate Division Problem [CLO92])**.** *To divide a polynomial $\theta(x,y)$ by several polynomials $\xi_1(x,y)$, ..., $\xi_n(x,y)$, we search n quotient polynomials $\gamma_1(x,y)$, ..., $\gamma_n(x,y)$ and a remainder polynomial $\epsilon(x,y)$ such that*

$$\theta(x,y) = \sum_{j=1}^{n} \gamma_j(x,y) \cdot \xi_j(x,y) + \epsilon(x,y), \tag{5.9}$$

and no monomial in $\epsilon(x,y)$ can be obtained by multiplying the leading monomial of any $\xi_j(x,y)$ with an element of Φ.

Of course, the division by one polynomial is just the special case $n = 1$. The following algorithmic description is adapted from [CLO92], but such a description can be found in almost any book on the basics of computational algebra because of the close relation to the calculation of Groebner bases. Note that while the description of the division uses a degree function like the one given in Definition 20, the existence of such a function is not a prerequisite for division to be possible. The use of this function is only to give a short notation for the condition that alignment has to be possible. Two important properties of a polynomial $f(x,y) = \sum_{j=0}^{i} f_j \varphi_j$ with degree $\rho(f) = \rho(\varphi_i)$ are its leading monomial $LM(f) = \varphi_i$ and the leading term $LT(f) = f_i \varphi_i$. They are needed to describe the division procedure and consequently for the decoding algorithm.

1. Set $j = 1$, $\gamma_1(x,y) = \cdots = \gamma_n(x,y) = \epsilon(x,y) = 0$.
2. Determine $i = \rho(\theta) - \rho(\xi_j)$.

3a. If there exists a monomial $\varphi(x,y) \in \Phi$ with $\rho(\varphi) = i$, then set $\gamma_j(x,y) = \gamma_j(x,y) + c\varphi(x,y)$ and $\theta(x,y) = \theta(x,y) - c\varphi(x,y)\xi_j(x,y)$, where the constant c is chosen such that $\rho(\theta)$ decreases, go to 4).

3b. If no such monomial exists and $j < n$ then set $j = j + 1$ and go to 2), if $j = n$ set $\epsilon(x,y) = \epsilon(x,y) + LT(\theta)$, $\theta(x,y) = \theta(x,y) - LT(\theta)$.

4. STOP if $\theta(x,y) = 0$, otherwise set $j = 1$ and go to 2).

Note that the result of such a division generally depends on the ordering of the divisors $\xi_1, \ldots, \xi_n$. While it is obvious for the quotients, e.g. if $LT(\theta) = xy$, $LT(\xi_1) = x$ and $LT(\xi_2) = y$, this is not so clear for the remainder. But to see that it can actually happen, consider the following example:

Example 17. Let $\theta(x,y) = xy+y+1$, $\xi_1 = xy+y$ and $\xi_2 = x$. In this ordering, the given procedure yields

$$\gamma_1 = 1, \gamma_2 = 0, \epsilon(x,y) = 1.$$

Interchanging the two divisors, i.e., $\bar{\xi}_1 = x$, $\bar{\xi}_2 = xy + y$, leads to

$$\bar{\gamma}_1 = y, \bar{\gamma}_2 = 0, \bar{\epsilon} = y + 1. \quad \bullet$$

Though this last example seems very fictitious, it should always be kept in mind that interchanging the order of the divisors may yield an entirely different result.

5.3 Solving the Key Equation with a Division Algorithm

In this section, we describe the actual algorithm that calculates a minimal solution to the key equation. It is similar in spirit to the decoding algorithm described by Sugiyama et al. [SKHN75] that was based on the extended Euclidean algorithm. As given by Theorem 11, for $t \leq \left\lfloor \frac{d-1}{2} \right\rfloor - s$, a minimal solution to the key equation is guaranteed to be a correct error locator, therefore we prove in the next section that the algorithm returns a minimal solution. Errors with weight $\left\lfloor \frac{d-1}{2} \right\rfloor - s < t \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ can be decoded with a small extension of the basic algorithm that is introduced in Section 5.5. For $t > \left\lfloor \frac{d-1}{2} \right\rfloor$, decoding is possible under certain circumstances and with major modifications to the basic algorithm, so this treatment is deferred to the next chapter.

Basic Idea of the Algorithm

To make sure that the algorithm always finds the solution with minimal $\rho(\Lambda)$, a series of trial polynomials $\Delta_i(x,y)$ with $LM(\Delta_i) = \varphi_i$ is constructed. Along with these polynomials, a second series of polynomials $R_i(x,y)$ is calculated in each iteration. Each pair of polynomials (Δ_i, R_i) is constructed in such a way that it fulfills (5.4) and $\rho(R_i)$ is minimal for a given $\rho(\Delta_i)$ (the latter statement is proved in the next chapter). By selecting the pair with smallest index i that also satisfies (5.5) as the final solution (Λ, R), the algorithm hence determines the solution with minimal $\rho(\Lambda)$. Because the algorithm stops as soon as (5.5) is fulfilled, we refer to this equation as the stopping criterion.

Initialization

As inputs to the algorithm, it is necessary to determine the syndrome elements and polynomial $S(x,y)$ from (5.1) and (5.2), as well as the polynomial y^{b_m+1} and the value ℓ needed in (5.4) and (5.5) respectively. For a consistent description set $\Delta_0(x,y) = 1$ and $R_0(x,y) = S(x,y)$ as the starting point. If the stopping criterion were already fulfilled for $\Delta_0(x,y)$ and $R_0(x,y)$, this would be equivalent to $S(x,y) = 0$ or the received word being a codeword. In such a case no decoding is necessary, so for the following steps one can assume that $S(x,y) \neq 0$.

Note that in the algorithm all operations are performed not only modulo the defining polynomial to stay in the given ring of polynomials, but also modulo the polynomial y^{b_m+1} even if the latter modulo operation is not explicitly mentioned.

Starting the Iterations

Searching for something alike the Euclidean algorithm, we want the polynomials to take the form

$$\begin{aligned} \Delta_i(x,y) &= \sum_{j<i} \gamma_{i,j}(x,y) \cdot \Delta_j(x,y), \\ R_i(x,y) &= \sum_{j<i} \gamma_{i,j}(x,y) \cdot R_j(x,y), \end{aligned} \tag{5.10}$$

with the number of summands as small as possible and the polynomials $\gamma_{i,j}$ obtained as quotient polynomials, R_i as the remainder of a division. Recall that the idea of the algorithm was to have $LM(\Delta_i) = \varphi_i$. Of course, it is in general not possible to have $\varphi_i = \varphi \cdot \varphi_{i-1}$ where $\varphi \in \Phi$, so it is necessary to take some polynomials from earlier iterations into account, too.

A special setup is used to ensure that $LT(\Delta_i) = \varphi_i$: choose $\Delta_{i_1}(x,y)$ so that $\varphi_{i_1} \cdot y = \varphi_i$. If this choice is not possible - this is the case whenever $\varphi_i = x^a$ - we pick Δ_{i_1} with $\varphi_{i_1} \cdot x = \varphi_i$. This procedure already determines the first term of $\gamma_{i,i_1}(x,y)$. The remaining terms as well as all other quotients $\gamma_{i,j}$ for $j < i$ are determined in the next step as the quotients of $\theta(x,y) = y \cdot R_{i_1}(x,y)$ (or $\theta(x,y) = x \cdot R_{i_1}(x,y)$ if $\varphi_i = x^a$) divided by all remainders $R_j(x,y)$ with $j < i$.

Although this setup looks different from the Euclidean algorithm, it is actually not: instead of choosing $\theta(x,y)$ as the dividend, one might search for a remainder R_{i_2} that has $\rho(R_{i_2}) = \rho(y) + \rho(R_{i_1})$, and divide this remainder by all other remainders. However, the next step shows that is easy to fix the leading monomial of $\Delta_i(x,y)$ with the construction described before, whereas this would be much harder to guarantee when choosing to divide R_{i_2}.

The Division Part of the Algorithm

As mentioned before, the result of a division depends on the ordering of the polynomials $R_j(x,y)$. We try those polynomials first where $\rho(\Delta_i)$ is largest, i.e., using the notation of Definition 24 we have $\xi_j = R_{i-j}$ for $j = 1, \ldots, i$. With this ordering we expect the number of polynomials involved in one division to be smaller than when they are tried in the order of calculation, however a specific ordering based on the degrees $\rho(R_j)$ can still yield a better performance. Further it is not necessary to always use all previous remainders in the division - details on which remainders need to be stored are given later. Compared to the division procedure given after Definition 24, an additional condition needs to be fulfilled: since we fixed the leading term of $\Delta_i(x,y)$ with the special setup in the previous step, now it becomes necessary to make sure that no polynomial $\gamma_{i,j}(x,y)$ is obtained that alters this leading term, i.e., one has to verify that

$$\rho(\Delta_j) + \rho(\gamma_{i,j}) < \rho(\varphi_i). \tag{5.11}$$

Because the $\gamma_{i,j}(x,y)$ are just the quotient polynomials obtained in the division procedure, it suffices to choose the monomial φ in Step 3 of the division

procedure from the set Φ_ν where $\nu = \rho(\varphi_i) - \rho(\Delta_j) - 1$. Once all quotient polynomials are determined, taking into account the leading term of γ_{i,i_1} obtained in the previous step, one can calculate the polynomial $\Delta_i(x, y)$ as given in (5.10).

Another modification to the basic division of bivariate polynomials is that we stop the division process as soon as one term in the intermediate dividend could not be cancelled. This significantly lowers the complexity, but does not change the result of the algorithm: the order of the remainder is the same in both cases and hence the decoder stops in the same iteration, but it is proven that any minimal solution to the key equation is unique given that the error weight is not too large, so an correct result is obtained in both cases.

This modification also allows to save time in other cases. Specifically, if there is no polynomial R_j with $j < i$ and $\rho(R_j) = \rho(\theta)$, then $\theta(x, y)$ would be the remainder polynomial of this iteration. But there is no difference in simply using the polynomials $y\Delta_{i_1}$ and yR_{i_1} (or $x\Delta_{i_1}$ and xR_{i_1}) respectively), so in such a situation, there is no need to actually store the result of this iteration. In Section 5.6, where the complexity of this algorithm is calculated, it is shown that this reduced number of polynomials also reduces the worst-case complexity.

The End of the Algorithm

In each iteration, check if the pair $\Delta_i(x, y)$, $R_i(x, y)$ satisfies the stopping criterion, i.e., if $\rho(R_i) - \rho(\Delta_i) \leq \ell$. If it is fulfilled, set

$$\Lambda(x, y) = \Delta_i(x, y) \text{ and } R(x, y) = R_i(x, y) \tag{5.12}$$

and stop the algorithm. Otherwise, increase i by one and perform another division. A summary of the algorithm for solving the key equation in pseudocode is given by Algorithm 1.

Algorithm 1: Solving the Key Equation for Hermitian Codes

Input: Polynomials $S \neq 0, y^{b_m+1}$; constant ℓ
Output: Locator polynomial Λ, evaluator polynomial R
Initialization: $i = 0$, $\Delta_0 = 1, R_0 = S$
repeat
  $i = i + 1$
  if $\varphi_i = x^a$ then $\varphi_{i_1} = x^{a-1}$, else $\varphi_{i_1} = \varphi_i / y$
  $\theta = \varphi_i / \varphi_{i_1} \cdot R_{i_1}$
  Divide θ by subset of $R_{i-1}, \ldots, R_0$: $\theta = \sum_j \gamma_{i,j} R_j + R_i$
  $\gamma_{i,i_1} = \gamma_{i,i_1} - \varphi_i / \varphi_{i_1}$
  $\Delta_i = -\sum_j \gamma_{i,j} \Delta_j$
until $\rho(\Delta_i) - \rho(R_i) \leq \ell$
$\Lambda = \Delta_i$, $R = R_i$

We conclude this section with an example that illustrates the functionality of the algorithm.

Example 18. Consider the Hermitian code $H(51)$ over $GF(4^2)$. This code has minimum distance $d = 13$, so it can correct all error patterns up to weight 5. Take the error word $\underline{\mathbf{e}} = (1, 1, 1, 1, 1, 0, \ldots, 0)$, i.e., the received word is in error

at the positions $(1, \alpha)$, $(1, \alpha^2)$, $(1, \alpha^4)$, $(1, \alpha^8)$, (α, α^6) and all error values are 1. The dual code has $m^\perp = 23$, so the syndrome polynomial is

$$\begin{aligned} S(x,y) =& x^4y^4 + \alpha x^3y^4 + \alpha^6 x^4y^3 + \alpha^2 x^2y^4 + \alpha^7 x^3y^3 + \alpha^{12} x^4y^2 + \alpha^3 xy^4 + \\ &+ \alpha^8 x^2y^3 + \alpha^{13} x^3y^2 + \alpha^{14} x^4y + \alpha^4 y^4 + \alpha^9 xy^3 + \alpha^{14} x^2y^2 + \alpha x^3y + \\ &+ \alpha^9 x^4 + \alpha^{10} y^3 + xy^2 + \alpha^{10} x^2y. \end{aligned}$$

From this the values $b_m = 4$ and $\ell = 12$ can be read off. In the first iteration, one finds $i_1 = 0$, and from this $\theta(x, y) = x \cdot S(x, y) \mod y^{b_m+1}$ (and also modulo the defining equation of the curve). $LT(\theta) = \alpha x^4 y^4$, so subtract $\alpha \cdot S(x, y)$ to obtain the remainder with

$$LM(R_1) = \alpha^4 x^4 y.$$

and $\Delta_1 = x + \alpha$. Because $\rho(R_1) - \rho(\Delta_1) = 21 - 4 = 17 > 12$, the algorithm is not finished yet. In the second iteration, one obtains

$$\Delta_2 = y + \alpha^2 x + \alpha^2 \text{ and } LM(R_2) = x^4 y^2,$$

and again $\rho(R_2) - \rho(\Delta_2) = 26 - 5 = 21 > 12$. In the third iteration,

$$\Delta_3 = x^2 + \alpha^4 x + \alpha \text{ and } LM(R_3) = \alpha^{10} y^4.$$

Now $\rho(R_3) - \rho(\Delta_3) = 20 - 8 = 12$, so the algorithm terminates. $\Delta_3(x, y)$ has 8 affine zeros on the curve:

$$(1, \alpha), (1, \alpha^2), (1, \alpha^4), (1, \alpha^8), (\alpha, \alpha^6), (\alpha, \alpha^7), (\alpha, \alpha^9), (\alpha, \alpha^{13}),$$

and this is a correct error locator because all error positions are contained in the set of zeros of $\Lambda(x, y) = \Delta_3(x, y)$. •

Note that in this case the correctness of the minimal solution was guaranteed. The entire remainder polynomials were not given here as the only important remainder in this example is $R_3(x, y) = R(x, y)$ if one wants to calculate the error values. However, this task has been addressed in many works (cf. [Ehr91], [PSP92], [OBA08] and others) so we do not further discuss it in this thesis.

5.4 Correctness of the Algorithm

Theorem 13. *The algorithm presented in the previous section always computes a minimal solution to the key equation.*

Proof. For each i, the algorithm calculates a pair (Δ_i, R_i) with $LM(\Delta_i) = \varphi_i(x, y)$ in increasing order. Each pair fulfills (5.4), and the pair with smallest index i that also fulfills (5.5) is returned. To show that the minimal solution is computed by the algorithm hence is equivalent to showing that $\rho(R_i)$ is as small as possible for any given $\rho(\varphi_i)$.

For those who are familiar with the matter, this last part is trivial because the steps of the algorithm are a Groebner-basis calculation (cf. [CLO92]). A more extensive proof uses Lemma 18 from Appendix B: if $\rho(R_i) = \rho_S - \rho(\varphi_{\bar{i}})$ there can be no polynomial $f(x, y)$, whether calculated in the algorithm or not, such that $\rho(f) < \rho(\Delta_i)$ and $\rho(f \cdot S) = \rho(R_i)$, but such a polynomial would be necessary to decrease $\rho(R_i)$ without changing $\rho(\Delta_i)$. Consequently there cannot be a pair (Δ_i', R_i') with $\rho(\Delta_i') = \rho(\Delta_i)$ but $\rho(R_i') < \rho(R_i)$ and the statement follows. □

Note that this theorem makes no statement about the correctness or uniqueness of the solution. These follow only for error weights within the bound given in Theorem 11 as the proof of the key equation shows. Therefore decoding failures are treated in the next section. These can occur if there is no unique minimal solution - then the algorithm returns any of the minimal solutions - or if there is a unique minimal solution that is not an error locator.

5.5 Handling of Decoding Failures

As indicated by the proof to Theorem 11, a solution pair (Λ, R) to the key equation is only guaranteed to be the correct error locator and evaluator polynomial if the error weight is bounded by some value $t < \lfloor \frac{d-1}{2} \rfloor$. On the other hand, it is a basic property of linear codes that *every* error with weight smaller than half the minimum distance can be uniquely mapped to a closest codeword, so it should be possible to find an extension to the algorithm that allows to decode these errors as well. In simulations we found that actually most errors with weight $t = \lfloor \frac{d-1}{2} \rfloor$ could also be corrected with the algorithm given so far without any modifications. Therefore, we want to have a criterion to identify these situations, and an extension to allow decoding in the other cases, too.

In order to decide if the polynomial $\Lambda(x, y)$ shall be accepted as the correct solution, it is first necessary to determine whether $\Lambda(x, y)$ actually is an error locator polynomial. But this is very simple as it just involves counting the number of zeros and comparing it with $\rho(\Lambda)$ as explained in Definition 23. This criterion sometimes also allows to identify errors with weight $t > \lfloor \frac{d-1}{2} \rfloor$: if $\rho(\Lambda) = \lfloor \frac{d-1}{2} \rfloor + g$, but the polynomial $\Lambda(x, y)$ returned from the algorithm has less than $\lfloor \frac{d-1}{2} \rfloor$ zeros one can say for sure that $t > \lfloor \frac{d-1}{2} \rfloor$. In such a case an extension like the one described in the following chapter is needed.

In case $\rho(\Lambda) < \lfloor \frac{d-1}{2} \rfloor + g$ but $\Lambda(x, y)$ has not enough zeros, decoding is possible with the following extension to the basic algorithm: first, a second pair (Δ_i, R_i) calculated in the algorithm is chosen, and then all linear combinations of those two polynomials are checked. Different criteria can be used to select one of these linear combinations as the final solution, these criteria and estimations of their performances are given later. The complexity of this extension is also derived in the next section.

But first consider the question how to select the second basis polynomial. Let the pair selected by Algorithm 1 be (Δ_i, R_i), then it is intuitive to just perform some more iterations of the algorithm until another pair (Δ_{i_1}, R_{i_1}) fulfilling (5.5) is found. However, keeping in mind that the aim is to find an error locator of minimal degree, it is possible to do better in some situations: before calculating a second pair, it is advisable to try all previously calculated pairs and check if $\rho(R_{i_1}) - \rho(\Delta_i) \leq \ell$ for some $i_1 < i$. If such a pair is found the final solution has $\rho(\Lambda) = \rho(\Delta_i)$, whereas by calculating a second pair one gets $\rho(\Lambda) > \rho(\Delta_i)$ and according to Theorem 11 the correct error locator shall have minimal degree. Either way, we obtain a set of candidates

$$\bar{\Lambda}(x, y) = \Delta_i(x, y) + \alpha_j \Delta_{i_1}(x, y).$$

Note that each time only one additional polynomial is selected, although there might be more $j \neq \{i, i_1\}$ with $\rho(R_j) - \rho(\bar{\Lambda}) \leq \ell$. This restriction is made to

keep the complexity of the algorithm small, but in simulations this was always sufficient to obtain a correct error locator as one of the candidates.

Second, a criterion is needed to decide which of the candidates to select as the final $\Lambda(x, y)$. Of course, one only needs to consider those candidates that actually are error locator polynomials, hence counting the number of zeros of each of the candidates is the first step that should always be made. This number of zeros also leads to the first criterion that was tested: among all candidates, the one with the largest number of zeros is selected. Simulations showed that this criterion already yields the correct solution in most cases (see the table at the end of the section). But while this first criterion is very simple, it can be improved if it is guaranteed that $t \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ errors occurred. A second criterion that was tested also searches the zeros of the corresponding polynomial

$$\bar{R}(x, y) = R_i(x, y) + \alpha_j R_{i_1}(x, y)$$

and discards all those solutions where the rational function $\bar{R}/\bar{\Lambda}$ has more than $\left\lfloor \frac{d-1}{2} \right\rfloor$ poles. This criterion yields a better result because any pole of $\bar{R}/\bar{\Lambda}$ leads to a nonzero coefficient in the reconstructed error word, but it is more complex because along with the zeros of two polynomials, their multiplicities have to be determined, too. Unfortunately these two criteria are still not able to correctly locate all error patterns of weight $\left\lfloor \frac{d-1}{2} \right\rfloor - s < t \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ for codes of any rate.

The third criterion that was investigated is to perform error evaluation for all candidates. Again it is a prerequisite that the error weight actually is smaller than half the minimum distance, then there can be only one solution for which the difference of the received word and reconstructed error yields a codeword. This follows immediately from the basic properties of linear codes, so this criterion always yields correct decoding. Unfortunately, the complexity of error evaluation is much larger than that of finding the zeros of a polynomial, so this third criterion should always be used only in combination with one of the other two criteria to limit the number of error evaluations performed.

To illustrate the extension, consider the following example that is slightly modified compared to Example 18:

Example 19. Take the same code as in Example 18 and the same error $\underline{\mathbf{e}}$ extended by another nonzero element in (α^2, α^3) also with error value 1. With this additional error $t = 6 = \left\lfloor \frac{d-1}{2} \right\rfloor > \left\lfloor \frac{d-1}{2} \right\rfloor - s$, so it is not guaranteed that the first solution obtained from the algorithm is a correct error locator (in fact, the error word for this example was specifically chosen to obtain such a situation). The syndrome polynomial of the modified error is

$$\begin{aligned}\tilde{S} = {} & \alpha^5 x^3 y^4 + \alpha^2 x^4 y^3 + \alpha^{10} x^2 y^4 + \alpha^{13} x^3 y^3 + \alpha^4 x^4 y^2 + \alpha^2 x y^4 \\ & + \alpha^{11} x^2 y^3 + \alpha^4 x^4 y + \alpha^5 y^4 + \alpha^{11} x^2 y^2 + \alpha^6 x^3 y + \alpha^8 x^4 \\ & + \alpha^{14} y^3 + \alpha^{11} x y^2 + \alpha^9 x^2 y.\end{aligned}$$

The polynomials Δ_i calculated in the first four iterations, as well as the degrees $\rho(\Delta_i)$ and $\rho(R_i)$ are given in Table 19. In the fourth iteration (5.5) is fulfilled for the first time. However, $\Delta_4(x, y)$ has only 3 affine zeros on the curve (see Example 15), so it is no error locator polynomial. Because

$$\rho(\Delta_4) = 9 < \left\lfloor \frac{d-1}{2} \right\rfloor + g = 12$$

i	Δ_i	$\rho(\Delta_i)$	$\rho(R_i)$
0	1	0	32
1	$x + \alpha^5$	4	36
2	$y + \alpha^{12}x + 1$	5	26
3	$x^2 + \alpha^5 x + \alpha^3$	8	21
4	$xy + \alpha^5 x^2 + y + \alpha^{13}x + \alpha^8$	9	20

Table 5.1: Polynomials calculated by Algorithm 1

the number of errors may be $t \leq \lfloor \frac{d-1}{2} \rfloor$ and the extension to the algorithm is used. Consequently, it becomes necessary to find a second basis polynomial. In this case, $\rho(R_3) - \rho(\Delta_4) = 21 - 9 = 12$, so there is no need to perform further iterations of the algorithm and (Δ_3, R_3) is chosen as the second basis pair. Forming all linear combinations of Δ_4 and Δ_3 and counting the number of zero positions, one finds that the linear combinations

$$\bar{\Lambda}_1 = \Delta_4 + \Delta_3 \text{ and } \bar{\Lambda}_2 = \Delta_4 + \alpha^{14}\Delta_3$$

both have 9 rational zeros and all other candidates have a smaller number of zeros (actually, there cannot be a polynomial $\bar{\Lambda}(x, y)$ with more zeros). The first criterion that counts only the number of zeros hence does not yield a unique result and a random decision has to be made. Since only the second option yields a correct error locator polynomial this decision may cause a wrong decoding. Using the extended criterion that counts the poles of $\bar{R}/\bar{\Lambda}$ shows that the first option belongs to an error pattern of weight $9 > \lfloor \frac{d-1}{2} \rfloor$, so this solution is discarded. Comparing the zeros of $\bar{\Lambda}_2$ and $\bar{R}_2$ shows that $\bar{R}_2/\bar{\Lambda}_2$ has only $6 = \lfloor \frac{d-1}{2} \rfloor$ poles, so this solution - which indeed constitutes the correct error locator - is chosen. •

Simulation Results

To demonstrate the functionality of the extension and the differences between the three acceptance criteria, a series of simulations was performed Hermitian codes with several design parameters m over $GF(4^2)$. With $q = 4$, (5.3) yields $s = 1$, so it is only necessary to simulate the decoding of errors with weight $t = \lfloor \frac{d-1}{2} \rfloor$. The design parameters, the resulting code rates $\frac{k}{n}$ and BMD decoding radius are given in the first three columns of Table 5.2. For each code, 10^7 random error patterns were used to test the algorithm. Because doing evaluation for all pairs identifies the correct solution for sure, only the other two criteria were tested, i.e., counting the zeros of Λ only and counting the poles[11] of R/Λ. The number E_{f1} in the fourth column gives the number of error patterns for which the solution returned by the basic algorithm was erroneously accepted due to the first criterion. The number N_{b1} shows the number of error words for which the extension was used, and the number N_{e1} denotes the number of errors

[11] A simplified version of this criterion was used: the distinct zeros of both R and Λ were counted and their number subtracted. This may lead to an estimated error weight that is too small (e.g. if P_i is a double zero of Λ and a simple zero of R, then it is also an error position) but because the criterion was used only to reject polynomials with too many poles a correct solution is never rejected.

where the first criterion lead to a wrong decision among the candidates (note that this number includes both random decisions and such where the correct error locator did not have the largest number of zeros). The numbers E_{f2}, N_{b2} and N_{e2} denote the results if the second criterion was used.

m	$\frac{k}{n}$	$\lfloor \frac{d-1}{2} \rfloor$	E_{f1}	N_{b1}	N_{e1}	E_{f2}	N_{b2}	N_{e2}
27	0.344	18	0	2034	0	0	2034	0
33	0.438	15	0	2050	0	0	2050	0
37	0.5	13	1	2170	1	1	2170	1
43	0.563	10	7	2018	3	7	2018	1
47	0.656	8	908	2814	383	572	3150	108

Table 5.2: Simulation Results for Several Codes $H(m)$

The table shows that the number of error patterns for which the extension had to be used does not differ much between the codes of different rates and is generally very low. In the last row, it can be seen that for the first criterion the extension is used even less, but this is due to the fact that more wrong solutions are erroneously accepted. The simulations show that even the simple criteria are able to identify the correct solution in a majority of these cases and for codes with low rates they, too, provide error-free correction.

5.6 Complexity of the Algorithm

In this section, we show that the asymptotic complexity of Algorithm 1 is $\mathcal{O}(n^{7/3})$, which is the same as that of other common decoding algorithms for AG codes. As the measure for complexity we count the average number of necessary multiplications. In the second part, we give the average complexity of the extension described in Section 5.5.

Complexity of the Basic Algorithm

Throughout this section, we use the notations introduced in Section 5.3, and let τ be the maximum number of correctable errors. Because the check matrix can be precalculated, the computation of the syndrome polynomial has complexity $\mathcal{O}(nm^{\perp})$. The selection of i_1 has linear complexity, and the calculation of θ is the multiplication of a polynomial by a monomial, this operation has complexity $\mathcal{O}(m^{\perp})$. Next, θ is divided by at most $\rho(\theta)$ other polynomials[12]. This division requires up to $\rho(\theta)\tau$ checks followed by $\mathcal{O}(\rho(\theta))$ subtractions of another polynomial, where a single subtraction has complexity $\mathcal{O}(m^{\perp})$ due to the multiplication of the divisor by a constant. Up to τ such divisions have to be performed, hence the overall complexity of this step is $\mathcal{O}(\rho(\theta)m^{\perp}\tau) = \mathcal{O}(n^3)$.

But closer investigation of the divisions shows that the overall complexity is actually smaller than this: First consider the case that $\rho(R_i) = \rho_S - \rho(\varphi_i)\ \forall\ i$. In this case, at most $2(q+1)$ subtractions are performed in any division, since

[12]Closer investigation of the divisions showed that $2(q+1)$ divisor polynomials are always sufficient. But since this part does not dominate the (asymptotic) complexity, we waive the proof and use the simpler estimate.

this number is equal to $\rho(\theta) - \rho(R_{i-1}) + 1$. If $\rho(R_i) < \rho_S - \rho(\varphi_i)$ for some i this observation is no longer true (more subtractions might be possible), but taking more than one iteration into account, this effect is reduced: as shown in Appendix B, a remainder with larger order is obtained eventually, and its calculation requires less subtractions. If $\rho(R_i) \ll \rho_S - \rho(\varphi_i)$ there may even be some remainders which need not be calculated at all. Though we cannot give a formal proof, simulations showed that the total number of subtractions in *all* iterations was $\mathcal{O}(\tau q)$, and with $q = n^{1/3}$ the overall average complexity becomes $\mathcal{O}(n^{7/3})$. The calculation of the polynomials Δ_i can be performed with the same complexity if it is performed in line with the division.

As stated in [Ehr91], the complexity of the evaluation step is $\mathcal{O}(n^2)$, hence the overall complexity of decoding is dominated by the basic algorithm which was $\mathcal{O}(n^{7/3})$. This complexity is smaller that of Shens subresultant algorithm [She92] which was estimated to be $\mathcal{O}(n^3)$.

Complexity of the Extension

Finally we want to compare the complexity of the presented algorithm with the "fast" algorithms from [JLJH92] and [SJM+95] that both have asymptotic complexity $\mathcal{O}(n^{7/3})$. But for a fair comparison, it is also necessary to estimate the complexity of the extension as the other algorithms, too, decode up to half the minimum distance, although in a different manner. Recall that in the extension two basis polynomials were selected so there exist q^2 candidate pairs. The two simple acceptance criteria that only count the number of zeros of these candidates both have overall complexity $q^2\mathcal{O}(n) = \mathcal{O}(n^{5/3})$ if the basis polynomials are evaluated at all points and the linear combinations are calculated in the "evaluation domain". From Table 5.2 it can be seen that this is sufficient for low-rate codes. For high-rate codes error evaluation for each candidate is necessary, and because the evaluation for a single pair (Λ, R) has complexity $\mathcal{O}(n^2)$ [Ehr91] the worst case complexity under this criterion is $\mathcal{O}(n^{8/3})$.

Though this worst case complexity is larger than that of the previous algorithms, two thing should be kept in mind when considering the practical impact of this result: first, this evaluation is only necessary for a small fraction of all decoded error words (in the simulations presented in Table 5.2 it was at most $3 \cdot 10^{-4}$), so one can expect a very small impact. Second, simulations showed that the number of candidates with an appropriate number of zeros according to one of the simple criteria is usually $\mathcal{O}(q)$ instead of $\mathcal{O}(q^2)$, so the *average* complexity of the extension using the evaluation criterion is $\mathcal{O}(n^{7/3})$.

Chapter 6

Extending Decoding Beyond Half the Minimum Distance

The decoding algorithm presented in the previous section is a BMD decoder, i.e., it can correct errors of weight up to half the minimum distance. For RS codes, there exist several BMD decoders and extensions for decoding beyond half the minimum distance. Some of these algorithms are designed for interleaved codes, where several codewords are transmitted in parallel and are corrupted by errors in the same positions. Other algorithms work for single codewords.

Most of the papers published on collaborative decoding of interleaved Reed-Solomon (IRS) codes, e.g. [BKY03], [KBB08], [SSB09], present extensions of well-known decoding algorithms for RS codes. For the mentioned papers, these are the Welch-Berlekamp, Sugiyama and Berlekamp-Massey algorithm respectively. On the other hand, interleaved Hermitian codes have never been a topic of interest to the community - the only work we are aware of that actually presents a decoding method is [BMS05], in [Arm08] that algorithm is compared to the decoding of IRS codes. However, that method extends the Welch-Berlekamp algorithm which is an interpolation-based method and hence quite different from Algorithm 1. In particular, no extension of an algorithm generalizing the Sugiyama algorithm has ever been presented.

Besides the works for interleaved codes, two different principles exist for the decoding of single RS codes beyond half the minimum distance. The first method is mainly used in the algorithms due to Sudan [Sud97] and Guruswami and Sudan [GS99], which are interpolation-based methods similar to the Welch-Berlekamp algorithm that return a list of all codewords lying within a certain distance from the received word. Another approach was described in [SSB06] and further studied in subsequent papers [SSB07], [SSB10]. It works on virtually extending the code into an interleaved RS code at the receiver side, this IRS code can then be decoded like any other IRS code. This kind of decoder declares a decoding failure if no unique solution does exist.

The original algorithm by Sudan [Sud97] has been extended to Hermitian codes in [HRN99] and to general AG codes in [SW99], the Guruswami-Sudan algorithm [GS99] on the other hand had been described for AG codes already in the original version. Opposed to this, no virtual extension method for AG

codes has been described yet. Due to the lack of works on interleaved Hermitian codes, this is not surprising. In this chapter, we give a decoding method based on the results of the previous chapter. For this purpose, we describe how to modify Algorithm 1 to obtain a basis for all solutions to the key equation and give the size of the basis. In Section 6.2, we discuss interleaved Hermitian codes and give an upper bound on the error weight such that an error can be uniquely located given the parameters of the interleaved code. Yet even if the bound is fulfilled, there is always a certain probability that unique decoding is not possible and the probability for this case is also given. We introduce virtual extension for Hermitian codes in Section 6.3 and give an upper bound on the code rate beyond which the use of virtual extension is not able to improve the maximum decoding radius.

6.1 A Basis for all Solutions

As mentioned, the algorithm can be used to correct beyond half the minimum distance, but larger modifications are necessary because there is always more than one solution for the key equation. The algorithm can be used to determine a basis for all those solutions, but the selection of a single solution requires more sophisticated methods. The reason is discussed at the end of this section. But first we describe the necessary modifications to the algorithm under the assumption that $t = \left\lfloor \frac{d-1}{2} \right\rfloor + t_0$ errors shall be corrected (with $t_0 > 0$) and then derive the number n_b of basis elements that is obtained.

In the original description of the algorithm, a stopping criterion was used to determine (along with the minimal error locator polynomial) an upper bound on the error weight of the received word. Unfortunately, no such stopping criterion exists if $t > \left\lfloor \frac{d-1}{2} \right\rfloor$. To be still able to correct, first fix a number t of errors that shall be corrected. Then use Algorithm 1 to calculate all pairs (Δ_i, R_i) with $\rho(\Delta_i) \leq \rho(\varphi_t)$. To do this, it is only necessary to replace the original stopping criterion ($\rho(R_i) - \rho(\Delta_i) \leq \ell$) by $\rho(\Delta_i) \leq \rho(\varphi_t)$. Lemma 12 also holds for $\rho(\Lambda) > \left\lfloor \frac{d-1}{2} \right\rfloor$, so those pairs with $\rho(R_i) > \rho(\varphi_t) + \ell$ are discarded. Before calculating the size of this basis, we give two important theorems.

Theorem 14. *The pairs (Δ_i, R_i) calculated by Algorithm 1 (without the stopping criterion) with $\rho(\Delta_i) \leq \rho(\varphi_t)$ and $\rho(R_i) \leq \rho(\varphi_t) + \ell$ form a basis for all solutions (of limited degree) to the key equation.*

Proof. The theorem is best proved by contradiction. Define the set

$$\Phi_b = \{LM(\Delta_i) | \rho(R_i) \leq \rho(\varphi_t) + \ell\},$$

and assume there is a solution (f, g) to the key equation with $\rho(f) \leq \rho(\varphi_t)$ that can be written as

$$f(x, y) = \sum_{i:\varphi_i \in \Phi_b} f_i \Delta_i(x, y) + f'(x, y)$$

with $f'(x, y) \neq 0$ and no monomial of $f'(x, y)$ is contained in Φ_b, in particular $LM(f') \notin \Phi_b$. By the definition of Φ_b and according to Lemma 18 (in Appendix B) there cannot be a polynomial with the given $\rho(f')$ and $\rho(f' \cdot S) \leq \rho(\varphi_t) + \ell$. With the linearity of polynomial addition this is a contradiction to the assumption that (f, g) is a solution to the key equation. □

Lemma 15. *Let the basis be selected as in Theorem 14. If t is large enough the error locator $\Lambda(x, y)$ can be expressed in terms of this basis as*

$$\Lambda(x, y) = \sum_{i:\varphi_i \in \Phi_b} \beta_i \Delta_i(x, y), \ \beta_i \in GF(q^2). \tag{6.1}$$

Proof. According to Lemma 12, a correct error locator is always a solution of the key equation, independent of $\rho(\Lambda)$. Because the selected basis generates all solutions to the key equation, the minimal error locator must be producible from the basis unless t was chosen too small: if the minimal error locator $\Lambda(x, y)$ has $\rho(\Lambda) = \mu$ but $\rho(\varphi_t) < \mu$, clearly $\Lambda(x, y)$ cannot be obtained from the given basis. □

Note that the coefficients β_i are allowed to be zero, so $\rho(\Lambda) < \rho(\varphi_t)$ is possible and the minimal error locator can always be obtained from the basis. On the other hand, any implementation of the algorithm requires some method to determine a sufficiently large value t. One could always use $t = d-1$ to be on the safe side but then the probability of obtaining many solutions (potentially even many correct solutions) is very high and a selection step is necessary.

To simplify matters, from now on we assume that the error weight t is known. To calculate the number of basis elements first assume that $\rho(R_i) = \rho_S - \rho(\varphi_i) \, \forall i$. Then the algorithm computes $|\Phi_{\rho(\varphi_t)}| = t+1$ pairs of polynomials, but some of them are not chosen for the basis: specifically, these are the pairs where

$$\rho(R_i) = \rho_S - \rho(\Delta_i) > \rho(\varphi_t) + \ell$$

or $\rho(\Delta_i) < m^\perp + 1 - \rho(\varphi_t)$. The number of such pairs is obtained in the same straightforward way, so the number of basis pairs is

$$n_b = |\Phi_{\rho(\varphi_t)}| - |\Phi_{m^\perp - \rho(\varphi_t)}| = t + 1 - |\Phi_{m^\perp - \rho(\varphi_t)}|. \tag{6.2}$$

To generalize this result to arbitrary $\rho(R_i)$, assume that there exists a pair $(\bar{\imath}, i)$ of indices where w.l.o.g. $\bar{\imath} < i$ and $\rho(R_{\bar{\imath}}) = \rho_s - \rho(\varphi_i)$. Close investigation of the polynomials obtained during the algorithm shows that then $\rho(R_i) = \rho_s - \rho(\varphi_{\bar{\imath}})$ (this is proved in Appendix B). To estimate the number of basis pairs under the changed circumstances consider the following three cases:

1. Both $\rho(R_i) \leq \rho(\varphi_t) + \ell$ and $\rho(R_{\bar{\imath}}) \leq \rho(\varphi_t) + \ell$, then both pairs are selected for the basis and n_b does not change. The situation is similar if both $\rho(R_i) > \rho(\varphi_t) + \ell$ and $\rho(R_{\bar{\imath}}) > \rho(\varphi_t) + \ell$, as then neither of the pairs is selected.

2. If $i < t$ and $\rho(R_i) > \rho(\varphi_t) + \ell$ but $\rho(R_{\bar{\imath}}) \leq \rho(\varphi_t) + \ell$ one can again calculate n_b by (6.2), but now the pair $(\Delta_{\bar{\imath}}, R_{\bar{\imath}})$ is picked instead of (Δ_i, R_i).

3. If $i > t$ then $\rho(R_{\bar{\imath}}) \leq \rho(\varphi_t) + \ell$ for sure, so the $\bar{\imath}$th pair is included in the basis. However, the pair (Δ_i, R_i) is not even calculated because of $\rho(\Delta_i)$. In this situation n_b is larger than indicated by (6.2), which turns out to be only a lower bound on the number of basis elements.

Although it might seem destructive that one can never be sure about the size of the basis, this last case does never contradict the results presented in this chapter. The reason is that beyond half the minimum distance unique decoding

can never be guaranteed, so this latter case only increases the probability that no unique decoding result can be obtained and is one of the reasons why no exact bounds on the failure probability can be given in the next section.

Unfortunately, the dependence of $|\Phi_m|$ on m is highly nonlinear for general m, so (6.2) can only be simplified in some special cases. The one special case we want to present here is $m^\perp - \rho(\varphi_t) > 2g-2$ and $t \geq g$ - this means that the error weight is neither too small nor too close to the minimum distance of the code. Under these conditions, the last term of (6.2) can be rewritten to

$$|\Phi_{m^\perp-\rho(\varphi_t)}| = m^\perp - \rho(\varphi_t) - g + 1 = m^\perp - (t+g) - g + 1,$$

and with this result

$$n_b = t + 1 - (m^\perp - t - 2g + 1) = 2t - (m^\perp - 2g + 2) + 2 = 2t - d + 2$$

because $m^\perp = d + 2g - 2$ for the codes we used. Recall the notation $t = \left\lfloor \frac{d-1}{2} \right\rfloor + t_0$ introduced at the beginning of the section, then

$$n_b = 2t_0 + 1. \tag{6.3}$$

This result coincides with known results for RS codes, see e.g. [SSB09], [KBB08].

Though theoretically it would be possible to use the same criteria as in Section 5.5 for all candidates obtained from the new basis, the complexity of this selection rapidly increases with an increasing error weight: the complexity increases by a factor of $q^2 = n^{2/3}$ for every additional basis element, so the complexity of the selection step is $\mathcal{O}(n^{2+4t_0/3})$ for $t = \left\lfloor \frac{d-1}{2} \right\rfloor + t_0$ and the specific case used in the derivation of (6.3). It might increase slower for other cases, but still the complexity is dominated by the selection. Therefore two approaches are presented in the next two sections that allow to reduce the number of possible solutions - if possible, a unique solution shall be obtained such that no selection is necessary. An entirely different option is the use of reliability information, but this idea is not considered in this thesis.

6.2 Interleaved Hermitian Codes

The use of interleaved codes increases the decoding radius compared to non-interleaved codes with a certain probability. The main reason why this is possible can be best explained with the help of Figure 6.1 that depicts an interleaved Hermitian (IH) code with η codewords of length n. It is possible to write a received word for each of the interleaved words, i.e.,

$$r^{(i)} = c^{(i)} + e^{(i)} \quad i = 1, \ldots, \eta,$$

and there is a separate key equation for each of the interleaved words so each of them can be decoded individually and this is still the best option if the errors occur randomly. But if the errors appear in a bursty pattern as depicted in Figure 6.1, where a burst corrupts one column of the array, collaboratively decoding the interleaved codewords reveals the strength of this construction: each of the codewords is corrupted in the same error positions (or a subset of all error positions). The error locator is the same under these circumstances and if the errors $e^{(i)}$ are linearly independent this allows to reduce the number

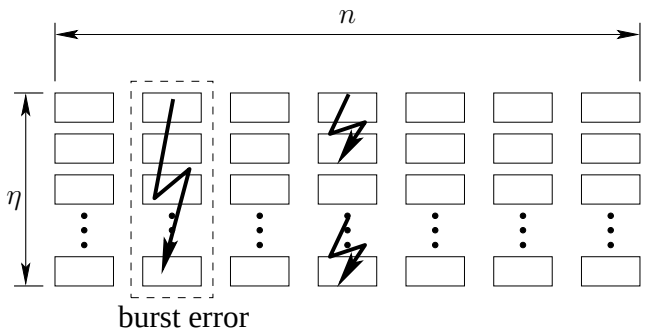


Figure 6.1: Interleaved Hermitian code and burst errors

of possible solutions. As long as the error weight is not too large, it is often possible to find a unique solution to all η key equations with high probability. The maximum decoding radius τ for which a unique solution can exist is derived in this section.

Generally, two types of interleaved codes are distinguished: *homogeneous* and *heterogeneous* IH codes. A code is called homogeneous if all η interleaved codewords belong to the same code. Otherwise, a code is called heterogeneous. Since homogeneous IH codes are a special case of heterogeneous codes, it suffices to consider the latter ones.

Basic Idea of Collaborative Decoding

The decoding scheme described here is maybe not optimal in terms of complexity, but it is simple to understand and allows a simple derivation of the desired bounds. The syndrome $S^{(1)}$ of the first error word is used to calculate a basis for all solutions of the key equation with the help of the algorithm from the previous section. All other syndromes are then used to reduce the number of possible solutions. The idea behind the reduction is that the same error locator Λ has to fulfill the key equation for all $S^{(j)}$, $j = 2, \ldots, \eta$, with respective polynomials $R^{(j)}$. As indicated by the index j, while $\Lambda(x, y)$ is the same for all errors, the remainders are (in general) different. If $\Lambda(x, y)$ is expressed in term of a basis with coefficients β_i, then each polynomial $R^{(j)}$ can be expressed in terms of the same coefficients β_i, and the coefficients of the monomials that have $\rho(\varphi_i) > \rho(\Lambda) + \ell^{(j)}$ (clearly the value ℓ depends on the design parameter m of a code, so it may be different for each codeword of an heterogeneous interleaved code) have to be zero.

The Number of Equations

We can use the coefficients of those $\varphi_{a,b}$ whose order is between $\rho(\Lambda) + \ell^{(j)} + 1$ and ρ_S and where $a \leq a_m$ and $b \leq b_m$. Counting shows that this number is

$$n_c^{(j)} = |\Phi_{\rho_S - \rho(R^{(j)})-1}| = |\Phi_{m^{\perp(j)} - \rho(\varphi_\tau)}|.$$

Here τ was used instead of t because the derivation is done for the maximum decoding radius τ instead of the actual error weight t. We only consider the

case that the maximum decoding radius is $\tau \geq g$, then $\rho(\varphi_\tau) = \tau + g$. Using the notation $\tau = \left\lfloor \frac{d^{(j)}-1}{2} \right\rfloor + \tau_0^{(j)}$ some manipulations lead to

$$m^{\perp(j)} - \rho(\varphi_\tau) = \ldots = \frac{d-1}{2} + \tau_0 + g - 2\tau_0 - 1 = \rho(\varphi_\tau) - 2\tau_0^{(j)} - 1 = \hat{m}^{(j)}.$$

If $\hat{m}^{(j)} > 2g-2$, one can use the known formula $|\Phi_m| = m - g + 1$. The number of equations obtained from one of the interleaved words depends on the number of errors and the designed minimum distance $d^{*(j)}$:

$$n_c^{(j)} = m^{\perp(j)} - (\tau + g) - g + 1 = d^{*(j)} - 1 - \tau \;\forall\, j = 2, \ldots, \eta, \tag{6.4}$$

and consequently the total number of equations is

$$n_c = \sum_{j=2}^{\eta} n_c^{(j)}. \tag{6.5}$$

One coefficient β_i can always be fixed by normalization (in compliance with the basic algorithm, $\Lambda(x, y)$ shall always be monic), so a unique solution can exist if $n_c \geq n_b - 1$. With the given formulas (6.5) and (6.2) for $t = \tau$, the maximum decoding radius is obtained:

$$\tau \leq \frac{\eta}{\eta + 1} \left(\frac{1}{\eta} \sum_{j=1}^{\eta} d^{*(j)} - 1 \right). \tag{6.6}$$

This formula shows that the number of correctable errors tends towards the average minimum distance (minus one) as the number of interleaved codewords increases. However, if $\tau \lesssim d^{*(j)}$, the assumption $\hat{m}^{(j)} > 2g - 2$ is no longer true and the maximum decoding radius is larger than the one given by (6.6): for $\hat{m}^{(j)} \leq 2g - 2$ write

$$|\Phi_{\hat{m}^{(j)}}| = \hat{m}^{(j)} - g + 1 + \sigma^{(j)}$$

with $\sigma^{(j)} > 0$. This leads to an increase in the maximum number of correctable errors by $\frac{\sigma^{(j)}}{\eta+1}$. The increased decoding radius is

$$\tau \leq \frac{\eta}{\eta + 1} \left(\frac{1}{\eta} \sum_{j=1}^{\eta} d^{*(j)} - 1 \right) + \sum_{j=1}^{\eta} \frac{\sigma^{(j)}(\tau)}{\eta + 1}. \tag{6.7}$$

The notation $\sigma^{(j)}(\tau)$ is used to emphasize the that $\sigma^{(j)}$ depends on τ. Yet it does so in a highly nonlinear way that cannot be expressed in a closed formula, so unfortunately it is not possible to solve (6.7) for τ. The best way to find the actual decoding radius is to use (6.6) to obtain an estimate, then increase τ step by step and always check if (6.7) is still fulfilled. Further, the result is only true as long as $\tau < d^{*(j)}$ for all interleaved codewords, because otherwise not all of the interleaved codewords can be reconstructed. Using this approach, the bound has been verified with a series of simulations.

Failure Probability

As mentioned before, even a sufficient number of equations does not guarantee that a unique decoding result can be obtained. We still assume that the error weight is known, so the correct error locator must be obtainable from the basis. Consequently the decoder will never return a wrong decoding result and it suffices to give an estimate of the probability of decoding failure, i.e., the probability that taking $S^{(2)}, \ldots, S^{(\eta)}$ into account does not suffice to obtain a unique result. The following statement holds for a channel model where each column is corrupted by a burst error with equal probability, so all burst error with a certain number t of corrupted positions are equally likely, but it is not necessary that within one burst all symbols are received erroneously.

Theorem 16 (Failure Probability)**.** *Take an interleaved Hermitian code over $GF(q^2) = GF(Q)$ consisting of η interleaved codes corrupted by errors in $t \geq g$ positions. Further let the values n_b and n_c be calculated according to* (6.2) *and* (6.4), (6.5) *respectively. Then the failure probability can be estimated by*

$$P_f \lesssim \frac{Q^{-(n_c - n_b + 1)}}{Q - 1}. \tag{6.8}$$

Note that this bound is much stronger than the one given in [BMS05]: translating the maximum decoding radius τ in (6.7) to their notation, their bound becomes $P_f < \bar{c}$ with $\bar{c} > 1$, which is a trivial statement. Yet (6.8) is very similar to the bound given in [SSB09, Theorem 7], so it is not surprising that the proof uses similar arguments. Note that the decoding problem is described a little differently for the proof: here, the special treatment of $S^{(1)}$ is no longer present, instead a large system of equations is solved that includes the syndrome matrices of all interleaved words. However, the aim is the same: finding a unique solution that solves all η key equations, and the question of uniqueness is independent on the actual computation.

Proof. The proof of [SSB09] is based on the decomposition of syndrome matrices $\underline{\mathbf{S}}^{(j)}$ (see Section 5.1) into matrix products

$$\underline{\mathbf{S}}^{(j)} = \underline{\mathbf{H}}^{(j)} \underline{\mathbf{F}}^{(j)} \underline{\mathbf{D}} \underline{\mathbf{V}},$$

where $\underline{\mathbf{D}}$ is a diagonal matrix with the error positions, $\underline{\mathbf{F}}^{(j)}$ a diagonal matrix with the error values and $\underline{\mathbf{V}}$ a Vandermonde matrix. It is essential that these three matrices are nonsingular.

For Hermitian codes, a similar decomposition is possible. The matrix $\underline{\mathbf{F}}^{(j)}$ has the same form as for RS codes, but the matrices $\underline{\mathbf{D}}$ and $\underline{\mathbf{V}}$ are different. While $\underline{\mathbf{D}}$ becomes an identity matrix (hence it can be dropped from the decomposition), $\underline{\mathbf{V}}$ now has evaluations of the $t+1$ bivariate monomials $\varphi_0, \ldots, \varphi_t, \in \Phi$ as entries:

$$\underline{\mathbf{V}} = \begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & x_1^3 & \cdots \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 & x_2^3 & \cdots \\ \vdots & & \vdots & & \vdots & & \vdots & \\ 1 & x_t & y_t & x_t^2 & x_t y_t & y_t^2 & x_t^3 & \cdots \end{pmatrix},$$

for simplicity the t error positions have been given indices $1, \ldots, t$. This matrix is not a Vandermonde matrix, and actually does not have full rank with a certain

probability p_r. Checking all possible combinations of error positions for small t and random combinations for larger t ($q = 4$ was fixed) showed that about 99.6% of these matrices $\underline{\mathbf{V}}$ have full rank, i.e., $p_r = 0.004$, so for the vast majority of cases the same arguments as in [SSB09] can be used. The discussion of the other cases is done afterwards.

If $\underline{\mathbf{V}}$ has full rank the proof works completely analogously to [SSB09, Theorem 7], the only difference is that $\underline{\mathbf{H}}^{(j)}$ now is a check matrix of a shortened Hermitian code of length t and dimension $t - \zeta^{(j)}$ instead of an RS code with the respective parameters, but this does not change the result of the proof as their main arguments hold for any q-ary linear code. The last relation that is independent of the kind of linear code is

$$P_f \leq \left(\frac{Q^{\eta+1-1}}{Q^\eta - 1}\right)^t \cdot \frac{Q^{-\sum_{j=1}^{\eta} \zeta^{(j)}}}{Q-1}.$$

For "sufficiently large" Q the first term can be approximated by $\left(\frac{Q^{\eta+1-1}}{Q^\eta-1}\right)^t \approx Q^t$. Note that this approximation already weakens the upper bound, but it is not the only reason why the theorem only states an approximate upper bound. For $j = 2, \ldots, \eta$, the number of rows of $\underline{\mathbf{H}}^{(j)}$ is

$$n_c^{(j)} = t - (t - \zeta^{(j)}) = \zeta^{(j)} = d^{(j)} - 1 - t.$$

For $j = 1$, (6.2) can be simplified to $n_b - 1 = 2t + 1 - d^{(1)} = t - \zeta^{(1)}$ under the given constraint. The statement follows by simple substitution.

Now turn to those cases where $\underline{\mathbf{V}}$ did not have full rank. Of course an upper bound on the failure probability is obtained if one assumes that decoding fails in all of these cases, i.e.,

$$P_f \lesssim p_r + \frac{Q^{-(n_c - n_b + 1)}}{Q-1}.$$

This bound is true under the following setup: t is the weight of the error, and the basis includes all pairs with $\rho(\Delta_i) \leq \varphi_t$ and $\rho(R_i) - \rho(\varphi_t) \leq \ell$. But sometimes there exist several solutions to the key equation with $\rho(\Lambda) \leq \rho(\varphi_t)$ (whether they are (correct) error locators or not does not matter), but the minimal error locator has $\rho(\Lambda) < \rho(\varphi_t)$. Of course, one can insist on using the obtained basis, and unter this setting the latter bound matched simulation results very well. However it is often possible to do better if in such a case the estimated error weight t is reduced and the failure probability is bounded again with the same arguments for this reduced value t. If necessary this reduction is performed several times and a failure is only declared if no nonzero solution was obtained after a reduction. These cases are already rare and in general $P_f(t) \gg P_f(t-1)$, because n_b is proportional to t, whereas n_c is reciprocal. This means that the failure probability is still close to the one calculated for the original value of t and the statement of the lemma follows. □

As mentioned in the calculation of n_b, the basis may be larger than the value n_b used here. On the other hand, simulations showed that this situation occurred in a few single occasions among millions of trials, and even if an increased basis was obtained this did not necessarily cause a decoding failure, so P_f was hardly increased.

k_1	k_2	k_2	t	$P_{f,b}$	$P_{f,s}$	t	$P_{f,b}$	$P_{f,s}$
22	28	32	23	$4.1 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	22	$6.3 \cdot 10^{-8}$	10^{-7}
28	32	38	19	$2.6 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$	18	$6.3 \cdot 10^{-8}$	$5 \cdot 10^{-7}$
32	38	42	16	$2.6 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$	15	$6.3 \cdot 10^{-8}$	$1.9 \cdot 10{-}5$

Table 6.1: Comparison of Theoretical and Actual Error Probability

The bound and its tightness are illustrated by some examples in Table 6.1. For the simulations, the correction of 10^7 error patterns was tried. For this simulations, the value t_0 was increased until a solution was found. Although this setup theoretically allows a wrong solution to be found, this never happened in any simulation so no error probabilities can be given. $P_{f,b}$ is the estimation for the failure probability given by (6.8), and $P_{f,s}$ denotes the probability that for the smallest t_0, for which a solution existed, this solution was not unique in the simulations. Note that the first two values in the last column correspond to 1 and 5 uncorrectable error patterns, so these values are not reliable enough to claim that the bound is wrong. The last value in the table is one where the bound is definitely not satisfied. But close investigation of the failures showed the same pattern in all 192 failure cases: though the decoding result was not unique, *all* solutions that were obtained corresponded to correct error locators. Clearly, selecting any of them would have led to a correct decoding result. Further, for 178 patterns another effect was seen: as the number of errors was quite close to the minimum distance of the first interleaved codeword, the correct error locator was already obtained from decoding the first syndrome. By including this check into the algorithm, i.e., to start the simulation with $t_0 = 0$, the failure rate could be immediately reduced to $1.4 \cdot 10^{-6}$.

Complexity of This Approach

This conceptually simple approach has cubic asymptotic complexity. Under the assumption that the number of errors is known, this is very simple to see: the complexity of finding the basis for all solutions is $\mathcal{O}(n^{7/3})$, because it is equal to the complexity of the basic algorithm. Setting up the linear system of equations requires $\eta - 1$ polynomial multiplications, each of these can be performed with at most quadratic complexity. However, the resulting linear system of equations has no specific structure, so finding the solution has (in general) complexity $\mathcal{O}(n^3)$.

The complexity does not increase even if the number of errors is unknown. However, that second case requires the use of structured methods for the solution of the linear system of equations. One example of such a method is the fundamental iterative algorithm (FIA) introduced in [FT91]: if two systems of equations shall be solved, where the first is a subsystem of the second one, then the solution of the first system can be reused to find the solution to the second system. In this way, both systems of equations can be solved with the same complexity as the larger system of equations.

In practice, the solution may be better if one wants to correct only a bit beyond half the (largest) minimum distance: if $\tau_0 \ll d$, this last step may easily have smaller complexity than finding the basis. However, there exist algorithms

for decoding IRS codes beyond half the minimum distance that have the same asymptotic complexity as those algorithms decoding regular codes up to half the minimum distance, e.g. [FT89] and [SSB09]. Because of all the correspondences between the decoding of Hermitian codes and RS codes, one may hope to find algorithms with smaller asymptotic complexity.

6.3 Virtual Extension to an Interleaved Code

The principle of virtual extension was first described in [SSB06] for RS codes, and the same principle can also be applied to Hermitian codes: at the receiver, new "received" words are formed by elementwise squaring the received word or raising it to higher powers, hence *virtually extending* the code into an interleaved code. Unfortunately, this principle provides a benefit only for codes with low rates, as is explained in the following paragraphs. The lower the code rate, the higher the powers that can be used. Here we restrict ourselves to giving the upper limit on the code rate for which virtual extension brings a benefit, i.e., for codes with higher rates the decoding radius does not increase if virtual extension is applied.

Recall that $r_j = c_j + e_j$, $j = 1, \ldots, n$, are the elements of the received word. By elementwise squaring, the elements of the "virtual" received word become

$$r_j^2 = (c_j + e_j)^2 = c_j^2 + 2c_j e_j + e_j^2 = f^2(P_j) + 2c_j e_j + e_j^2. \tag{6.9}$$

Define $\underline{\mathbf{r}}^{<2>} = \underline{\mathbf{c}}^{<2>} + \underline{\mathbf{e}}^{<2>}$ with the new codeword $\underline{\mathbf{c}}^{<2>}$ and the new error $\underline{\mathbf{e}}^{<2>}$ where

$$e_j^{<2>} = 2c_j e_j + e_j^2.$$

Because the elements of the new error word depend only on the error and codeword at the same position, $\underline{\mathbf{e}}$ and $\underline{\mathbf{e}}^{<2>}$ are located by the same error locator polynomial. The elements of the new codeword are obtained by evaluating $f^2(x, y)$ instead of $f(x, y)$ and (in the same way as for univariate polynomials) this means that the degree of the polynomial doubles. If the original codeword belongs to the code $H(m)$, then the new codeword $\underline{\mathbf{c}}^{<2>}$ belongs to the code $H(2m)$, so a heterogeneous IH code is generated. The maximum decoding radius for this kind of code was given in (6.6), with $d^{(1)} = n - m$ and $d^{(2)} = n - 2m$ the bound becomes

$$\tau \leq \frac{2}{3}(n-1) - m.$$

Of course the effort of virtual extension is in vain if the decoding radius is not increased, i.e., if $\tau \leq \left\lfloor \frac{d-1}{2} \right\rfloor$. This yields the upper bound

$$m < \frac{n-1}{3},$$

so if $m \geq \frac{n-1}{3}$ there is no improvement in the number of correctable errors. While the bound on m looks familiar if one knows the corresponding results for RS codes [SSB06], one should keep in mind that m is not the dimension of the code. To derive an upper bound on the code rate $\frac{k}{n}$, it would be great if one could use the fact $k = m - g + 1$ if $m > 2g - 2$. To see that it is allowed to use

Number of "interleaved" words η	2	3	4	5	6
Bound on m	$\frac{n-1}{3}$	$\frac{n-1}{6}$	$\frac{n-1}{10}$	$\frac{n-1}{15}$	$\frac{n-1}{21}$

Table 6.2: Upper Bound on the Design Parameter for Virtual Extension

that formula consider the upper bound $m = \left\lfloor \frac{n-2}{3} \right\rfloor$. Comparing this value to the "critical value" $2g - 2$ shows that

$$m = \left\lfloor \frac{n-2}{3} \right\rfloor \approx \frac{q^3 - 2}{3} > 2g - 2 = q^2 - q - 2 \;\Leftrightarrow\; q > 2,$$

so the formula may be used for all applicable values of q: for $q = 2$ one actually finds that for both $m = 1$ and $m = 2$ (these are the only values of m which are small enough) the maximum correction radius obtained by virtual extension fulfills $\tau \not\geq \left\lfloor \frac{d-1}{2} \right\rfloor + 1$, that means that the decoding radius is theoretically increased, but because the error weight is always an integer it is not possible to correct more errors than before. For all other q, this leads to the rate bound

$$\frac{k}{n} \leq \frac{1}{3} - \frac{1}{3n}(3g + 2). \tag{6.10}$$

For $q \to \infty$, this bound is $R \leq \frac{1}{3}$ and hence coincides with the rate restriction for RS codes. For small q, however, the restriction is more severe.

At the beginning of the section, we stated that above this upper bound virtual extension is useless. To show this, we repeat the previous steps, this time comparing the virtual extension that uses a squared word only to one where the elements of the received word are also raised to the third power, i.e.,

$$\begin{aligned} \underline{\mathbf{r}}^{<3>} = \underline{\mathbf{c}}^{<3>} + \underline{\mathbf{e}}^{<3>} \text{ with } r_j^3 = r_j^{<3>} &= c_j^3 + 3c_j^2 e_j + 3c_j e_j^2 + e_j^3 \\ \text{and } c_j^{<3>} = c_j^3, e_j^{<3>} &= 3c_j^2 e_j + 3c_j e_j^2 + e_j^3 \end{aligned}$$

and so an interleaved code with $\eta = 3$ is obtained[13]. Again, the computation of this additional "received" word is only useful if the decoding radius is increased, i.e., if

$$\frac{3}{4}(n-1) - \frac{3}{2}m > \frac{2}{3}(n-1) - m,$$

and this is fulfilled if

$$m < \frac{1}{6}(n-1),$$

i.e., the restriction is even stronger. For the general case of η interleaved codewords belonging to the codes $H(m), H(2m), \ldots, H(\eta m)$, the upper bound on m is

$$m < \frac{2}{\eta(\eta+1)}(n-1).$$

Table 6.2 lists the bounds for some values η. Because Hermitian codes have $g > 0$, the actual rate restriction is even more severe than the bound on m. Yet the bound on m coincides with the results for RS codes obtained in [SSB06]. Further it is usually necessary to verify if the increase in the decoding radius is large enough that the code can actually correct at least one error more (a counterexample is the case $q = 2$ discussed before).

[13] Using only the original received word $\underline{\mathbf{r}}$ and $\underline{\mathbf{r}}^{<3>}$ does not make sense as this IH code can correct less errors than the one considered before because of the larger value $m^{<3>} = 3m$.

Chapter 7

Conclusions

In this thesis, we presented a new decoding algorithm for Hermitian codes. It uses repeated divisions of bivariate polynomials, similar to the extended Euclidean algorithm used for the decoding of Reed-Solomon codes. In this description it is different to previously published decoding algorithms for Hermitian codes, but achieves the same decoding radius with the same complexity as the fastest known algorithms. Our algorithm calculates and returns only a single error locator polynomial, whereas many other locator decoding algorithms calculate a basis for the locator ideal. It hence presents an alternative to previously published algorithms.

Another new development presented in this thesis is locator decoding beyond half the minimum distance. The conceptually simplest setting in which decoding beyond half the minimum distance is possible uses interleaved codes. These codes have been neglected by most of the community so far, though extending the existing decoding algorithms to this setting is rather simple as had been shown in Sections 6.1 and 6.2. Compared to the only existing work on interleaved codes [BMS05], the method presented here achieves a larger maximum decoding radius and for a certain error weight also a smaller probability of decoding failure.

In contrast, several works present interpolation-based methods for decoding non-interleaved Hermitian codes beyond half the minimum distance, e.g. [GS99], [HRN99], [SW99]. All these algorithms are list decoders, i.e., they return a list of all codewords lying within a certain distance from the received word, but a syndrome-based method like the one from [SSB06], that finds the unique closest codeword with high probability, had never been considered. For locator decoding, the idea is to reduce the problem to one of decoding an interleaved code by creating "virtual" interleaved words at the receiver. We show at the end of the thesis that this extension is possible in the same way it was done for RS codes and give the maximum code rate for which this decoding provides a benefit.

Another main difference to other papers is that this thesis was written for people with moderate previous knowledge about algebra or algebraic geometry: an introduction to algebraic geometry is given, restricted to those topics that are essential in the definition of an AG code. The topics are often not introduced

in their most general form, but restricted to special cases (e.g. defining only the affine and projective line and plane instead of general n-dimensional spaces), or used constructive instead of formal definitions (e.g. in the definition of local parameters). This also helps to keep things as simple as possible. Further, almost all definitions and theorems are illustrated with examples. Although a general definition of AG codes is given, the actual decoding algorithm is derived only for the special subclass of Hermitian codes.

The entire algorithm is stated in terms of bivariate polynomials and therefore avoids the necessity to study more advanced algebraic objects such as differential forms. This description of the algorithm allows to understand it from a separate definition of Hermitian codes that uses algebra only to the extent necessary to describe RS codes, and to apply the algorithm and its extension without extensive studies of algebraic geometry. In order to build a bridge to other works on AG codes, we also give a short introduction to algebraic geometry and define both RS and Hermitian codes as a special case of algebraic geometric codes.

Hopefully, this approach (especially the alternative definition of Hermitian codes, also used in the papers in which parts of this work had been presented before) will help to bring the interest of the coding community (back) to AG codes: although the codes and a lot of decoding algorithms had been presented already fifteen to twenty or even more years ago, there are still no applications that utilize these codes. One reason might be that most works on AG codes had been written by mathematicians, and either they did not contain specific decoding algorithms or the description used many results from algebraic geometry making the result hard to understand for engineers. But as shown in this thesis, these codes are not much more difficult to understand and to use than Reed-Solomon codes. This small drawback is definitely compensated by the fact that Hermitian codes do not suffer from the same severe length restriction induced by the ground field: over the field $GF(q^2)$, the maximum length of an RS code is q^2 whereas it is q^3 for a Hermitian code; a significant increase in the code length that promises improved performance. We think that there is no reason why Hermitian codes should not be applied and that the time has come for engineers to consider Hermitian codes when designing new applications.

Outlook

However, this thesis is far from presenting the end to research on this topic - and history shows that research on a certain code class was often rather pushed than slowed down by emerging applications using those codes, even though the focus might change. In the present case, several improvements to the algorithms are still possible. For the basic algorithm, it seems probable that improvements in the asymptotic complexity will come from more efficient methods for basic operations like polynomial multiplications: many researchers have already tried to find BMD decoders for Hermitian codes, but still a complexity of $\mathcal{O}(n^{7/3})$ is the best ever found. Another possibility would be the use of an algorithm similar to the so-called fast Euclidean algorithm, which can significantly improve the decoding complexity for RS codes, but it is not yet known if such an algorithm exists for Hermitian codes as well. Smaller improvements may arise from an intensive study of the remainders used in each iteration, e.g. it may be possible not to calculate certain remainders because they are not necessary. While it is

not sure if these reductions can decrease the asymptotic complexity, they should not be neglected in applications.

In contrast, research has only begun on decoding algorithms for interleaved Hermitian codes: the main purpose of Chapter 6 was to derive the maximum decoding radius for interleaved codes and to show that an algorithm which is capable of decoding up to this radius (with high probability) does exist. While the complexity of this algorithm is polynomial in the code length, it is larger than the complexity of the algorithm that achieves decoding up to half the minimum distance. For RS codes, there exist algorithms that achieve the larger decoding radius with the same complexity (e.g. [SSB09], [SSB10], or [FT89] for the homogeneous case only) so one may hope to find a decoding algorithm for interleaved codes that has the same complexity as the basic algorithm and still achieves the same increased radius.

Beyond these improvements, a generalization of the decoding algorithm to other kinds of AG codes would be interesting. Another approach which we did not consider yet for Hermitian codes is to include reliability information about the received word in the decoding process. An interpolation based method to do so had been presented in [LO10], yet again the comparison to Reed-Solomon codes stirs expectations that an efficient use of reliability information in a locator decoding scheme should also be possible.

Appendix A

Further Valuations on Hermitian Curves

In this section, we first illustrate the meaning of tangent (unfortunately, this cannot be done for finite fields) and then derive the valuation of a function

$$f(x,y,z) = \frac{(x+\alpha_i z)^a (y+\beta_i z)^b}{z^{a+b}}$$

in the point $(\alpha_i, \beta_i, 1)$ on the Hermitian curve over $GF(4^2)$.

The Tangent to a Curve

Although the meaning of tangent cannot be visualized over the finite field, for two special points - namely $(0,0)$ and $(0,-1)$ - it is possible to get a visualization by plotting the defining polynomial on the affine plane over the real numbers, as is done in Figure A.1.

One can see that the x-axis (given by $y=0$) is tangent to the curve in $(0,0)$ and a parallel line to it ($y=-1$) is tangent to the curve in $(0,-1)$, whereas the y-axis ($x=0$) intersects the curve in both these points. This means that the function x can be used as a local parameter in both points. Unfortunately, the exact valuation (multiplicity) of $y=0$ or $y=-1$ respectively cannot be derived from the figure, even if only a smaller part of the curve were drawn. For this, a formal calculation of the valuation is necessary.

More Valuations

As mentioned, we give the valuation of the function

$$f(x,y,z) = \frac{(x+\alpha_i z)^a (y+\beta_i z)^b}{z^{a+b}}$$

in $Q = (\alpha_i, \beta_i, 1)$. First, take any point Q with $\alpha_i \neq 0$, and consequently also $\beta_i \neq 0$. In these points

$$\tau_Q = \alpha_i^4 x + y + \beta_i^4 z.$$

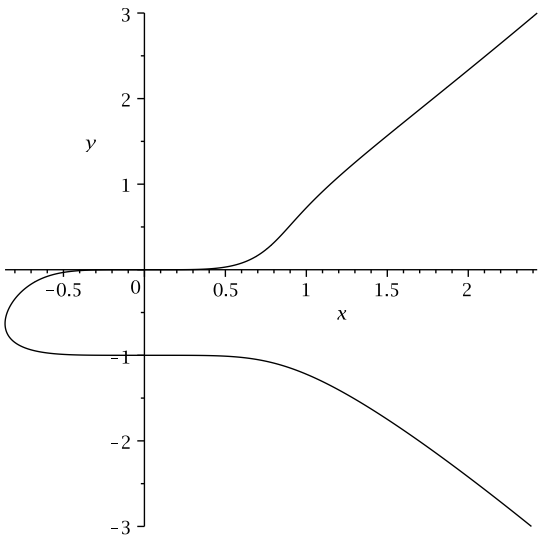


Figure A.1: Hermitian curve $x^5 - y^4 - y = 0$ over $\mathbb{R}$

Both the functions $\frac{x+\alpha_i z}{z}$ and $\frac{y+\beta_i z}{z}$ are possible local parameters, because neither $x + \alpha_i z$ not $y + \beta_i z$ is a multiple of τ_Q so with (3.6)

$$v_Q\left(\frac{(x+\alpha_i z)^a (y+\beta_i z)^b}{z^{a+b}}\right) = a + b.$$

Next, consider the point $Q = P_0 = (0, 0, 1)$. In this point

$$f(x, y, z) = \frac{x^a y^b}{z^{a+b}},$$

and the tangent is $\tau_Q = y$, so a possible local parameter is $t_Q = \frac{x}{z}$ (compare this with the result over the real numbers given before). It hence remains to determine the valuation of $\frac{y}{z}$. This function can be rewritten as follows:

$$\frac{y}{z} = \frac{x^5}{z(z^4 + zy^3)} = \frac{x^5}{z^5} \cdot \frac{z^3}{z^3 + y^3}.$$

The evaluation

$$\left.\frac{z^3}{z^3 + y^3}\right|_{P_0} = 1$$

immediately gives the valuation $v_Q\left(\frac{y}{z}\right) = 5$, and the wanted result is

$$v_{P_0}\left(\frac{x^a y^b}{z^{a+b}}\right) = a + 5b.$$

Finally, consider a point with $\alpha_i = 0$ but $\beta_i \neq 0$. The tangent in these points is $\tau_Q = y + \beta_i^4 z = y + \beta_i z$ (the latter equality is only valid in the points where $\alpha_i = 0$), so again $\frac{x}{z}$ is a possible local parameter. As an intermediate step, first calculate the valuation of

$$\tilde{f}(x, y, z) = \frac{y + \beta_i z}{z} = \frac{y}{z} + \beta_i.$$

This valuation is best calculated with the help of Lemma 5: $\tilde{f}(x, y)$ can only have a pole if $z = 0$. On a Hermitian curve the only such point is $P = (0, 1, 0)$, and the given function indeed does have a pole in P, so it is possible to use (3.8) and the result of Example 9 to find

$$v_P(\tilde{f}) = \min\{v_P\left(\frac{y}{z}\right), v_P(\beta_i)\} = \min\{-5, 0\} = -5,$$

where the first equality holds because the valuations of $\frac{y}{z}$ and β_i are distinct. Note that it was not possible to directly use (3.8) in $Q = (0, \beta_i, 1)$, because both $v_Q\left(\frac{y}{z}\right) = 0$ and $v_Q(\beta_i) = 0$, so (3.8) only gives $v_Q(\tilde{f}) \geq 0$. This is an even weaker statement than $v_Q(\tilde{f}) > 0$ which follows immediately from the fact that $\tilde{f}(Q) = 0$.

In all other points, $\tilde{f}(x, y) = 0 \Leftrightarrow y = \beta_i$. Taking a look at Figure 3.2 one sees that Q is the only point on $\mathcal{X}$ at which $\tilde{f}(x, y)$ can be zero. Now Lemma 5 implies that

$$v_Q\left(\frac{y + \beta_i z}{z}\right) = 5.$$

With this result, it follows that

$$v_Q\left(f(x, y, z)\right) = v_Q\left(\frac{x^a(y + \beta_i z)^b}{z^{a+b}}\right) = a + 5b,$$

so the valuation in a point depends only on the respective value α_i but not on β_i.

Appendix B

Degrees of the Remainder Polynomials

In this appendix, we give the proofs of two important lemmas that are used in the proof of correctness of the basic algorithm and in the calculation of the number of basis elements n_b.

Lemma 17. *Consider two iterations $\bar{\imath} \neq i$ of the division decoding algorithm of Chapter 5. If $\rho(R_{\bar{\imath}}) = \rho_S - \rho(\varphi_i)$, then the reverse relation*

$$\rho(R_i) = \rho_S - \rho(\varphi_{\bar{\imath}}),$$

always holds.

Proof. To derive the result, use the matrix form of the key equation. The upper left corner of the syndrome matrix $\underline{\mathbf{S}}$ is

$$\underline{\mathbf{S}} = \begin{pmatrix} s_{0,0} & s_{1,0} & s_{0,1} & s_{2,0} & s_{1,1} & s_{0,2} & \cdots \\ s_{1,0} & s_{2,0} & s_{1,1} & s_{3,0} & s_{2,1} & s_{1,2} & \cdots \\ s_{0,1} & s_{1,1} & s_{0,2} & s_{2,1} & s_{1,2} & s_{0,3} & \cdots \\ s_{2,0} & s_{3,0} & s_{2,1} & s_{4,0} & s_{3,1} & s_{2,2} & \cdots \\ s_{1,1} & s_{2,1} & s_{1,2} & s_{3,1} & s_{2,2} & s_{1,3} & \cdots \\ s_{0,2} & s_{1,2} & s_{0,3} & s_{2,2} & s_{1,3} & s_{0,4} & \cdots \\ \vdots & & \vdots & & \vdots & & \ddots \end{pmatrix}.$$

Denote by $\underline{\mathbf{S}}_{\bar{\imath},i}$ the submatrix of $\underline{\mathbf{S}}$ consisting of the first $\bar{\imath}+1$ rows and $i+1$ columns of $\underline{\mathbf{S}}$. Recalling the relation to the polynomial key equation, the fact that

$$\rho(R_{\bar{\imath}}) = \rho_S - \rho(\varphi_i)$$

can be restated in terms of matrices: it is equivalent to the fact that $\underline{\mathbf{S}}_{\bar{\imath}-1,i}$ does not have full rank (consequently, all matrices $\underline{\mathbf{S}}_{j,i}$ with $j < \bar{\imath}$ do not have full rank), but the matrix $\underline{\mathbf{S}}_{\bar{\imath},i}$ does have full rank.

To show that $\rho(R_i) = \rho_S - \rho(\varphi_{\bar{\imath}})$, first note that the matrices $\underline{\mathbf{S}}_{i,j}$ for $j < \bar{\imath}$ cannot have full rank because these are just the transpose of the matrices $\underline{\mathbf{S}}_{j,i}$ with $j < \bar{\imath}$. This means that for these j the degrees of the remainders must be

$\rho(R_j) \neq \rho_S - \rho(\varphi_{\bar{\imath}})$ because one of the conditions for equivalence is not fulfilled. On the other hand, $\underline{\mathbf{S}}_{i,\bar{\imath}}$ does have full rank, so it remains to show that $\underline{\mathbf{S}}_{j,\bar{\imath}}$ for $0 \leq j < i$ does not have full rank. For this purpose, define the value $\bar{\jmath}$ to be the smallest integer for which the matrix $\underline{\mathbf{S}}_{\bar{\jmath},j}$ has full rank. Then there are two possibilities:

1. $\bar{\jmath} > \bar{\imath}$. Repeating the arguments of the last paragraph for $\bar{\jmath}$ instead of $\bar{\imath}$ shows that $\underline{\mathbf{S}}_{j,\bar{\imath}}$ cannot have full rank.

2. $\bar{\jmath} < \bar{\imath}$. This means that a remainder with $\rho(R_{\bar{\jmath}}) = \rho_S - \rho(R_j)$ already exists, but by construction two remainders cannot have the same degree. This is equivalent to the fact that $\underline{\mathbf{S}}_{j,\bar{\imath}}$ cannot have full rank.

Note that the case $\bar{\jmath} = \bar{\imath}$ needs not be considered because it would contradict the definition of $\bar{\imath}$ in the statement of the Lemma. □

Another implication of the equivalence between the rank of a submatrix of $\underline{\mathbf{S}}$ and the degree of a remainder is the following lemma which is not only concerned with the polynomials obtained from the algorithm, but arbitrary polynomials of restricted degree.

Lemma 18. *Let (Δ_i, R_i) be a pair of polynomials calculated by Algorithm 1. Then there does not exist a polynomial $f(x, y)$ with $\rho(f) < \rho(\Delta_i)$ and $\rho(fS) = \rho(R_i)$.*

Proof. The degree condition could only be fulfilled if $\underline{\mathbf{S}}_{i,j}$ would have full rank for some $j < \bar{\imath}$, but this is not possible as the previous proof shows. □

This latter lemma now implies that the degree of a remainder calculated by Algorithm 1 is as small as possible.

Bibliography

[Arm08] Marc A. Armand. Interleaved Reed-Solomon Codes versus Interleaved Hermitian Codes. *IEEE Communications Letters*, 12(10):779 –781, October 2008.

[Ber68] Elwyn R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968.

[BK] Irene I. Bouw and Sabine Kampf. Decoding Hermitian Codes with a Division Algorithm. Submitted to *Advances in Mathematics of Communications*.

[BKY03] Daniel Bleichenbacher, Aggelos Kiayias, and Moti Yung. Decoding of Interleaved Reed Solomon Codes over Noisy Data. In Jos C.M. Baeten, Jan K. Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2003.

[Bla03] Richard E. Blahut. *Algebraic Codes for Data Transmission*. Cambridge University Press, 2003.

[Bla08] Richard E. Blahut. *Algebraic Codes on Lines, Planes and Curves*. Cambridge University Press, 2008.

[BMS05] Andrew Brown, Lorenz Minder, and M. Amin Shokrollahi. Improved Decoding of Interleaved AG Codes. In Nigel Smart, editor, *Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 37–46. Springer, 2005.

[Bos99] Martin Bossert. *Kanalcodierung*. Teubner Verlag, 1999.

[CLO92] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties and Algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 1992.

[Duu08] Iwan M. Duursma. Algebraic Geometry Codes: General Theory. In Edgar Martínez-Moro, Carlos Munuera, and Diego Ruano, editors, *Advances in Algebraic Geometry Codes*. World Scientific Publishing Co., 2008.

[Ehr91] Dirk Ehrhard. *Über das Dekodieren algebraisch-geometrischer Codes.* Heinrich-Heine-Universität Düsseldorf, 1991.

[FT89] Gui-Liang Feng and Kenneth K. Tzeng. A Generalized Euclidean Algorithm for Multisequence Shift-Register Synthesis. *IEEE Transactions on Information Theory*, 35(3):584 –594, May 1989.

[FT91] Gui-Liang Feng and Kenneth K. Tzeng. A Generalization of the Berlekamp-Massey Algorithm for Multisequence Shift-Register Synthesis with Applications to Decoding Cyclic Codes. *IEEE Transactions on Information Theory*, 37:1274–1287, September 1991.

[Giu] Massimo Giulietti. Notes on Algebraic Geometric Codes. Online. *http://www.math.kth.se/math/forskningsrapporter/Giulietti.pdf.*

[Gop83] Valerii D. Goppa. Algebraic-Geometric Codes. *Mathematics of the USSR-Izvestiya*, 21(1):75, 1983.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[HLP98] Tom Høholdt, Jacobus H. van Lint, and Ruud Pellikaan. Algebraic Geometry Codes. In Vera S. Pless, W. Cary Huffman, and Richard A. Brualdi, editors, *Handbook of Coding Theory*, volume I, pages 871–961. Elsevier, Amsterdam, 1998.

[HP95] Tom Høholdt and Ruud Pellikaan. On the Decoding of Algebraic-Geometric Codes. *IEEE Transactions on Information Theory*, 41(6):1589 –1614, November 1995.

[HRN99] Tom Høholdt and Rasmus Refslund Nielsen. Decoding Hermitian Codes with Sudan's Algorithm. In Marc Fossorier, Hideki Imai, Shu Lin, and Alain Poli, editors, *Proceedings of AAECC-13*, volume 1719 of *Lecture Notes in Computer Science*, pages 260–270. Springer, 1999.

[JH04] Jørn Justesen and Tom Høholdt. *A Course in Error-Correcting Codes (EMS Textbooks in Mathematics).* European Mathematical Society, February 2004.

[JLJH92] Jørn Justesen, Knud J. Larsen, Helge E. Jensen, and Tom Høholdt. Fast Decoding of Codes from Algebraic Plane Curves. *IEEE Transactions on Information Theory*, 38(1):111 –119, January 1992.

[KB10] Sabine Kampf and Martin Bossert. The Euclidean Algorithm for Generalized Minimum Distance Decoding of Reed-Solomon Codes. In *IEEE Information Theory Workshop 2010*, August 2010.

[KBB08] Sabine Kampf, Martin Bossert, and Sergey Bezzateev. Some Results on List Decoding of Interleaved Reed-Solomon Codes with the Extended Euclidean Algorithm. In *Proc. Coding Theory Days in St. Petersburg*, pages 31–36, St. Petersburg, Russia, October 2008.

[Köt96] Ralf Kötter. Fast Generalized Minimum Distance Decoding of Algebraic-Geometry and Reed-Solomon Codes. *IEEE Transactions on Information Theory*, 42(3):721–737, May 1996.

[LG88] Jacobus H. van Lint and Gerard van der Geer. *Introduction to Coding Theory and Algebraic Geometry*. Birkhäuser, 1988.

[Lin90] Jacobus H. van Lint. Algebraic Geometric Codes. In Dijen Ray-Chaudhuri, editor, *Coding Theory and Design Theory, Part I*, pages 137–162. Springer, 1990. The IMA Volumes in Mathematics and its Applications, Volume 20.

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, October 1996.

[LO10] Kwankyu Lee and Michael E. O'Sullivan. Algebraic Soft-Decision Decoding of Hermitian Codes. *IEEE Transactions on Information Theory*, 56(6):2587 –2600, June 2010.

[Mas69] James L. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, 15(1):122 – 127, January 1969.

[MS88] F. Jessie MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes (North-Holland Mathematical Library)*. North Holland, June 1988.

[OBA08] Michael E. O'Sullivan and Maria Bras-Amorós. The Key Equation for One-Point Codes. In Edgar Martínez-Moro, Carlos Munuera, and Diego Ruano, editors, *Advances in Algebraic Geometry Codes*. World Scientific Publishing Co., 2008.

[Por88] Sidney C. Porter. *Decoding Codes Arising from Goppa's Construction on Algebraic Curves*. Yale University, 1988.

[PSP92] Sidney C. Porter, Ba-Zhong Shen, and Ruud Pellikaan. Decoding Geometric Goppa Codes Using an Extra Place. *IEEE Transactions on Information Theory*, 38(6):1663–1676, November 1992.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

[Sha48] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[She92] Ba-Zhong Shen. Solving a Congruence on a Graded Algebra by a Subresultant Sequence and its Application. *Journal of Symbolic Computation*, 14(5):505–522, 1992.

[SJM+95] Shajiro Sakata, Jørn Justesen, Y. Madelung, Helge E. Jensen, and Tom Høholdt. Fast Decoding of Algebraic-Geometric Codes up to the Designed Minimum Distance. *IEEE Transactions on Information Theory*, 41(6):1672 –1677, November 1995.

[SKHN75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A Method for Solving Key Equation for Decoding Goppa Codes. *Information and Control*, 27(1):87–99, 1975.

[Sor93] Ulrich K. Sorger. A new Reed-Solomon Code Decoding Algorithm Based on Newton's Interpolation. *IEEE Transactions on Information Theory*, 39(2):358 –365, March 1993.

[SSB06] Georg Schmidt, Vladimir R. Sidorenko, and Martin Bossert. Decoding Reed–Solomon Codes Beyond Half the Minimum Distance using Shift-Register Synthesis. In *IEEE International Symposium on Information Theory*, pages 459–463, Seattle, WA, USA, July 2006.

[SSB07] Georg Schmidt, Vladimir R. Sidorenko, and Martin Bossert. Enhancing the Correcting Radius of Interleaved Reed-Solomon Decoding using Syndrome Extension Techniques. In *IEEE International Symposium on Information Theory*, pages 1341–1345, Nice, France, June 2007.

[SSB09] Georg Schmidt, Vladimir R. Sidorenko, and Martin Bossert. Collaborative Decoding of Interleaved Reed-Solomon Codes and Concatenated Code Designs. *IEEE Transactions on Information Theory*, 55(7):2991 –3012, July 2009.

[SSB10] Georg Schmidt, Vladimir R. Sidorenko, and Martin Bossert. Syndrome Decoding of Reed–Solomon Codes Beyond Half the Minimum Distance Based on Shift-Register Synthesis. *IEEE Transactions on Information Theory*, 56(10):5245 – 5252, October 2010.

[SSBZ10] Christian Senger, Vladimir R. Sidorenko, Martin Bossert, and Victor V. Zyablov. Multitrial decoding of concatenated codes using fixed thresholds. *Problems of Information Transmission*, 46(2):127–141, June 2010.

[Sud97] Madhu Sudan. Decoding of Reed-Solomon Codes beyond the Error-Correction Bound. *Journal of Complexity*, 13(1):180–193, March 1997.

[SW99] M. Amin Shokrollahi and Hal Wasserman. List Decoding of Algebraic-Geometric Codes. *IEEE Transactions on Information Theory*, 45(2):432–437, March 1999.

[WB86] Loyd R. Welch and Elwyn R. Berlekamp. Error Correction for Algebraic Block Codes. US Patent 4 633 470, December 1986.

List of Publications

Publications containing parts of this thesis

- [BK] Irene I. Bouw, Sabine Kampf, *"Syndrome decoding for Hermite codes with a Groebner bases algorithm"*, submitted to *Advances in Mathematics of Communications*
- Sabine Kampf, *"Bounds on Collaborative Decoding of Interleaved Hermitian Codes and Virtual Extension"*, submitted to *3ICMCTA special issue of Designs, Codes and Cryptography*
- Sabine Kampf, *"Bounds on Collaborative Decoding of Interleaved Hermitian Codes with a Division Algorithm and Virtual Extension"*, 3rd International Castle Meeting on Coding Theory and Applications 2011(3ICMCTA), Cardona, Spain
- Sabine Kampf, Martin Bossert and Irene I. Bouw, *"Solving the Key Equation for Hermitian Codes with a Division Algorithm"*, IEEE International Symposium on Information Theory 2011, St. Petersburg, Russia

Further publications

- [KB10] Sabine Kampf and Martin Bossert, *"The Euclidean Algorithm for Generalized Minimum Distance Decoding of Reed-Solomon Codes"*, IEEE Information Theory Workshop 2010, Dublin, Ireland
- Sabine Kampf and Martin Bossert, *"A Fast Generalized Minimum Distance Decoder for Reed-Solomon Codes Based on the Extended Euclidean Algorithm"*, IEEE International Symposium on Information Theory 2010, Austin, TX, USA
- Sabine Kampf, Antonia Wachter and Martin Bossert, *"A Method for Soft-Decision Decoding of Reed-Solomon Codes Based on the Extended Euclidean Algorithm"*, International ITG Conference on Source and Channel Coding (SCC) 2010, Siegen, Germany
- Alexander Zeh, Sabine Kampf and Martin Bossert, *"On the Equivalence of Sudan-Decoding and Decoding via Virtual Extension to an Interleaved Reed-Solomon Code"*, International ITG Conference on Source and Channel Coding (SCC) 2010, Siegen, Germany
- [KBB08] Sabine Kampf, Martin Bossert and Sergey Bezzateev, *"Some Results on List Decoding of Interleaved Reed-Solomon Codes with the Extended Euclidean Algorithm"*, Workshop "Coding Theory Days in St. Petersburg" 2008, St. Petersburg, Russia

The CV is not included in the online version for reasons of data protection.

Der Lebenslauf ist in der Online-Version aus Gründen des Datenschutzes nicht enthalten.