3.8 Goppa-Code

V.C. Goppa hat im Jahr 1970 einen Code veröffentlicht, der als Verallgemeinerung des BCH-Codes angesehen werden kann (Originalaufsatz in russischer Sprache: *V.C. Goppa*, "A New Class of Linear Error-Correcting Codes", Probleme Peredatshi Information (=Titel der Zeitschrift), Band 6, Seiten 24-30, Jahrgang 1970). Seine praktische Bedeutung für die Fehlerkorrektur ist – zur Zeit wenigstens – noch gering. Da das Verfahren zur Codeerzeugung aber eine interessante Ergänzung der schon bekannten Techniken darstellt und es auch für die später zu behandelnden Verschlüsselungsmethoden Bedeutung hat, ist ein Blick auf das Arbeitsprinzip nützlich. Im übrigen kann man den BCH-Code als zyklischen Spezialfall des im allgemeinen nicht-zyklischen Goppa-Codes betrachten.

3.8.1 Erzeugung der Codewörter

Sehen wir uns zuerst an, wie man die Codewörter erzeugt, und beschränken uns dazu auf den binären Fall. Die Codelänge entspricht beim Goppa-Code der Anzahl von Elementen im gewählten Erweiterungskörper $GF(2^m)$. Ein Codewort v hat also 2^m Elemente und kann z. B. als n-Tupel geschrieben werden:

$$v = (v_1\, v_2\, v_3\, \ldots\ldots\, v_n)$$

Konkurrenz für BCH?

Wenn man wie gewohnt die höchste zu korrigierende Fehlerzahl mit t bezeichnet, so besteht die Grundidee darin, 2^m verschiedene Polynome $h_i(z)$ über $GF(2^m)$ mit dem Grad t-1 zu finden und jedes einzelne mit einem solchen v_i zu multiplizieren, dass die Summe

$$\sum_{i=1}^{n=2^m} v_i \cdot h_i(z) \;=\; v_1 \cdot h_1(z) + v_2 \cdot h_2(z) + \ldots\ldots + v_n \cdot h_n(z)$$

den Wert Null annimmt. Da die Polynome $h_i(z)$ den Grad t-1 haben, weist jedes einzelne davon genau „t“ Koeffizienten auf:

$$h_i(z) = h_{i,t-1} \cdot z^{t-1} + h_{i,t-2} \cdot z^{t-2} + \ldots\ldots + h_{i,1} \cdot z + h_{i,0}\ .$$

Für ein Beispiel wird das Galoisfeld $GF(2^4)$ zugrunde ge-

legt, wobei die Codewörter die Länge n = 16 haben. Wenn man t=3 Fehler korrigieren können will, müssen 16 Polynome 2.Grades über GF(2^4) gefunden werden. Natürlich sind nur ganz bestimmte Polynome dafür geeignet und Goppa hat hierzu Berechnungsvorschriften angegeben. Wir nehmen uns diese Vorschriften etwas später vor und geben hier zunächst nur die ersten vier an (zur Erinnerung: die Koeffizienten β, β^2 usw. des GF(2^4) und deren Struktur findet man in der Tabelle 3-20 auf Seite 145):

$$h_1(z) = z^2 + 1$$
$$h_2(z) = z^2 + z$$
$$h_3(z) = \beta^8 z^2 + \beta^9 z + \beta$$
$$h_4(z) = \beta z^2 + \beta^3 z + \beta^2$$
$$\cdot \quad \cdot \quad \cdot$$

Summe cum Laude

Der obige Summenausdruck für die Polynome $h_i(z)$ lässt sich auch als

$$\left(\sum_{i=1}^{n} v_i \cdot h_{i,t-1}\right) \cdot z^{t-1} + \left(\sum_{i=1}^{n} v_i \cdot h_{i,t-2}\right) \cdot z^{t-2} + \ldots + \left(\sum_{i=1}^{n} v_i \cdot h_{i,1}\right) \cdot z + \sum_{i=1}^{n} v_i \cdot h_{i,0} = 0$$

schreiben. Der Summenausdruck ist also selbst ein Polynom in z vom Grad t-1, welches in allen Koeffizienten die Codewortelemente v_i enthält. Ein Polynom kann unabhängig von z nur dann den Wert Null annehmen, wenn alle Koeffizienten verschwinden. Mit einer entsprechenden Wahl der Codewortelemente v_1, v_2, ..., v_n lässt sich das erreichen. Da „t“ Koeffizienten vorhanden sind, stehen t Gleichungen zur Verfügung, denen die Codewortelemente genügen müssen. Im vorliegenden Beispiel sind das 3, die sich mit Hilfe der Tabelle 3-16 zusammenstellen lassen. Für den Koeffizienten bei z^2 gilt

Parität im Koeffizientenpelz

$$\begin{aligned} &v_1 + v_2 + \beta^8 v_3 + \beta v_4 + \beta^{11} v_5 + \beta^2 v_6 \\ &+ \beta^{10} v_7 + \beta^7 v_8 + \beta^{10} v_9 + \beta^4 v_{10} + \beta^{13} v_{11} \\ &+ \beta^5 v_{12} + \beta^5 v_{13} + \beta^{14} v_{14} + \beta^{10} v_{15} + \beta^5 v_{16} = 0 \end{aligned}$$

für den Koeffizienten bei z^1

$$0v_1 + v_2 + \beta^9 v_3 + \beta^3 v_4 + \beta^{14} v_5 + \beta^6 v_6$$
$$+ v_7 + \beta^{13} v_8 + \beta^2 v_9 + \beta^{12} v_{10} + \beta^7 v_{11}$$
$$\bullet \quad v_{12} + \beta v_{13} + \beta^{11} v_{14} + \beta^8 v_{15} + \beta^4 v_{16} = 0,$$

und für den Koeffizienten bei $z^0 = 1$ gilt schließlich

$$v_1 + 0v_2 + \beta v_3 + \beta^2 v_4 + \beta^9 v_5 + \beta^4 v_6$$
$$+ v_7 + \beta^3 v_8 + \beta^{13} v_9 + \beta^8 v_{10} + \beta^{12} v_{11}$$
$$\bullet \quad v_{12} + \beta^{14} v_{13} + \beta^6 v_{14} + \beta^7 v_{15} + \beta^{11} v_{16} = 0 .$$

Diese 3 Gleichungen sind nichts anderes als die bereits vom Hamming-Code her bekannten Paritätsgleichungen. Codewörter stellen demnach alle diejenigen Kombinationen der v_i dar, für welche die Paritätsgleichungen erfüllt sind. Damit liegt ein lineares Gleichungssystem zur Bestimmung der Codewort-Elemente v_i vor. Alle Rechenoperationen werden im GF(2^4) ausgeführt. Da die Informationen aber als Bits erscheinen, ist es wie schon beim Reed-Solomon-Code zweckmäßig, die binäre Polynomdarstellung für die β-Elemente gemäß Bild 3.17 zu verwenden:

$$\begin{vmatrix} 0000 & 1001 & 0010 & 0100 \\ 0010 & 1110 & 1011 & 1011 \\ 0001 & 1011 & 1101 & 1011 \\ 1110 & 0011 & 1110 & 0110 \\ 0011 & 1101 & 0110 & 0100 \\ 0000 & 0101 & 1100 & 0110 \\ 0010 & 0000 & 0110 & 1101 \\ 0100 & 1011 & 0111 & 0011 \\ 0000 & 1001 & 1010 & 1111 \\ 0001 & 0000 & 1110 & 0101 \\ 0010 & 1100 & 0010 & 0011 \\ 1000 & 0110 & 1111 & 1010 \end{vmatrix} \cdot \begin{vmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \\ v_{13} \\ v_{14} \\ v_{15} \\ v_{16} \end{vmatrix} = H * v^T = 0$$

Bild 3.17: *Paritätsgleichung eines 16-stelligen Goppa-Codes im GF(2^4) in Matrizen-Schreibweise*

Wege durch's Dickicht

Wie erhält man hieraus eine Rechenvorschrift zur Bestimmung der Codewörter und wie viele Codewörter sind es? Antworten auf diese Fragen geben die mathematischen Standardverfahren. Nach Bild 3.17 liegen zur Bestimmung der 16 Codewortelemente v_i insgesamt 12 Gleichungen vor, d. h., unser Gleichungssystem ist un-

terbestimmt. Denn mit 12 Gleichungen lassen sich natürlich immer nur 12 Unbekannte berechnen. Allerdings kann man die restlichen 4 willkürlich festlegen, sie sind dann eben *nicht* unbekannt.

Nutzlast-Transporter

Genau das ist aber notwendig, da man auf jeden Fall *auch* frei wählbare Informationsanteile im Code unterbringen muss. Damit steht folgendes fest: Von den 16 Variablen können 4 vorgegeben werden, mit denen sich $2^4 = 16$ Informationswörter erzeugen lassen. Das ist zugleich die Anzahl der Codewörter.

Jetzt besteht die Aufgabe noch darin, diese Codewörter zu berechnen (sie werden übrigens für t=3 einen Mindesthammingabstand von d=7 haben). Um das Gleichungssystem in Bild 3.17 zu lösen, bringt man es zunächst auf obere Dreiecksform. Dazu wird aus der Matrix H durch elementare Zeilenoperationen MOD 2 eine obere Dreiecks-ähnliche Form erzeugt, die in Bild 3.18 mit H' bezeichnet ist (der Berechnungsvorgang ist an sich sehr einfach, es empfiehlt sich aber wegen der verhältnismäßig großen Zahl von Teilschritten, diese mühsame Arbeit einem Programm zu überlassen).

$$\left|\begin{array}{cc} 1110\ 0011\ 1110\ 0110 & v_1 \\ 0100\ 1011\ 0111\ 0011 & v_2 \\ 0010\ 1110\ 1011\ 1011 & v_3 \\ 0001\ 1011\ 1101\ 1011 & v_4 \\ & v_5 \\ 0000\ 1001\ 0010\ 0100 & v_6 \\ 0000\ 0101\ 1100\ 0110 & v_7 \\ 0000\ 0010\ 1001\ 1000 & v_8 \\ 0000\ 0001\ 0010\ 0000 & v_9 \\ & v_{10} \\ 0000\ 0000\ 1101\ 0100 & v_{11} \\ 0000\ 0000\ 0101\ 1111 & v_{12} \\ 0000\ 0000\ 0010\ 0111 & v_{13} \\ 0000\ 0000\ 0000\ 1001 & v_{14} \\ & v_{15} \\ & v_{16} \end{array}\right| = H' * v^T = 0$$

Bild 3.18: *veränderte Paritätsgleichung gemäß Bild 3.17 auf obere Dreiecks-ähnliche Form*

Auf Biegen und Vertauschen

Wenn jetzt noch die Spalten 12 und 13 vertauscht werden, erhält man die echte obere Dreiecksmatrix H'' in Bild 3.19. Man beachte, dass durch den Spaltentausch auch die Codewortelemente v_{12} und v_{13} ihre Plätze wechseln und der Codewortvektor v in v' übergeht.

Jetzt nimmt man die Elemente v_{12}, v_{14}, v_{15} und v_{16} als die 4 unabhängigen, frei wählbaren Informationsanteile. Dadurch werden für die Bildung der 12•12-Systemmatrix des Gleichungssystems in der Matrix H'' die 4 letzten Spalten abgetrennt und auf die rechte Seite geschafft.

$$\left|\begin{array}{l} 1110\ 0011\ 1110\ 0110 \\ 0100\ 1011\ 0110\ 1011 \\ 0010\ 1110\ 1011\ 1011 \\ 0001\ 1011\ 1101\ 1011 \\ \\ 0000\ 1001\ 0010\ 0100 \\ 0000\ 0101\ 1100\ 0110 \\ 0000\ 0010\ 1001\ 1000 \\ 0000\ 0001\ 0010\ 0000 \\ \\ 0000\ 0000\ 1100\ 1100 \\ 0000\ 0000\ 0101\ 1111 \\ 0000\ 0000\ 0010\ 0111 \\ 0000\ 0000\ 0001\ 0001 \end{array}\right| \left|\begin{array}{l} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{13} \\ v_{12} \\ v_{14} \\ v_{15} \\ v_{16} \end{array}\right| = H'' * v'^{T} = 0$$

Bild 3.19: *Paritätsgleichung in oberer Dreiecksmatrixform*

$$\left|\begin{array}{l} 0110 \\ 1010 \\ 1101 \\ 1110 \\ \\ 0100 \\ 1101 \\ 1011 \\ 0111 \\ \\ 0010 \\ 1110 \\ 0111 \\ 0001 \\ \\ 1000 \\ 0100 \\ 0010 \\ 0001 \end{array}\right| \left|\begin{array}{l} v_{12} \\ v_{14} \\ v_{15} \\ v_{16} \end{array}\right| = G'^{T} * vi^{T} = \left|\begin{array}{l} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{13} \\ v_{12} \\ v_{14} \\ v_{15} \\ v_{16} \end{array}\right|$$

Bild 3.20: *Matrizengleichung zur Erzeugung eines Goppa-Codes im GF(2^4)*

von hinten her aufrollen

Das durch den Rest, nämlich durch die ersten 12 Spalten, gebildete Gleichungssystem lässt sich jetzt von der letzten Zeile her schrittweise lösen. Als Ergebnis erhält man die Generatormatrix G' in Bild 3.20, aus der sich durch Multiplikation mit den Informationsvektoren vi die 16 Codewortvektoren bilden lassen. Das Verfahren ist bereits vom Hamming-Code her bekannt. In Bild 3.21 sind die Codewörter v' aufgelistet. Man beachte, dass nach wie vor die Elemente v_{12} und v_{13} vertauscht sind.

```
v'(0)  = [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
v'(1)  = [ 0 0 1 0 0 1 1 1 0 0 1 1 0 0 0 1 ]
v'(2)  = [ 1 1 0 1 0 0 1 1 1 1 1 0 0 0 1 0 ]
v'(3)  = [ 1 1 1 1 0 1 0 0 1 1 0 1 0 0 1 1 ]
v'(4)  = [ 1 0 1 1 1 1 0 1 0 1 1 0 0 1 0 0 ]
v'(5)  = [ 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 ]
v'(6)  = [ 0 1 1 0 1 1 1 0 1 0 0 0 0 1 1 0 ]
v'(7)  = [ 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 ]
v'(8)  = [ 0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 0 ]
v'(9)  = [ 0 1 0 1 0 0 0 1 0 1 1 1 1 0 0 1 ]
v'(10) = [ 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 ]
v'(11) = [ 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 ]
v'(12) = [ 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 ]
v'(13) = [ 1 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 ]
v'(14) = [ 0 0 0 1 1 0 0 0 1 1 0 0 1 1 1 0 ]
v'(15) = [ 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ]
```

Bild 3.21: *Die 16 Codewörter des Goppa-Codes im GF(2^4)*

```
| 0000 1001 0010 0100 |  | 0011 1100 0011 1100 |
| 0010 1110 1011 1011 |  | 0011 0011 1100 1100 |
| 0001 1011 1101 1011 |  | 0101 1010 1010 0101 |
| 1110 0011 1110 0110 |  | 0011 1100 1100 0011 |
|                     |  |                     |
| 0011 1101 0110 0100 |  | 0000 1111 0000 1111 |
| 0000 0101 1100 0110 |  | 0101 1010 1010 0101 |
| 0010 0000 0111 0101 |  | 0110 0110 1001 1001 |
| 0100 1011 0110 1011 |  | 0110 1001 0110 1001 |
|                     |  |                     |
| 0000 1001 1011 0111 |  | 0011 0011 0011 0011 |
| 0001 0000 1110 0101 |  | 0011 1100 1100 0011 |
| 0010 1100 0010 0011 |  | 0110 1001 0110 1001 |
| 1000 0110 1111 1010 |  | 0101 0101 0101 0101 |
                         |                     |
                         | 0000 0000 1111 1111 |
                         | 0000 1111 0000 1111 |
                         | 0011 0011 0011 0011 |
                         | 0101 0101 0101 0101 |

= H* * V'^T =

| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
|                     |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
|                     |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
| 0000 0000 0000 0000 |
```

Bild 3.22: *Paritätskontrolle der 16 Codewörter*

Zur Kontrolle stellt man durch Multiplikation aller Codewörter mit der Paritätsprüfmatrix H* (bei der gegenüber

Schon Lenin wusste: Vertrauen ist gut, Mißtrauen ist besser

H die Spalten 12 und 13 vertauscht wurden) fest, dass tatsächlich jedesmal der Nullvektor herauskommt. In Bild 3.22 wurden dazu die Codewortvektoren aus Bild 3.21 zur transponierten Matrix V'^T zusammengefasst.

Genau genommen weiß man damit allerdings nur, dass man das Gleichungssystem richtig gelöst hat. Um festzustellen, dass die Codewörter auch wirklich einen Mindest-Hammingabstand von d=7 aufweisen, berechnet man, wie in Unterkapitel 3.7.5 für zyklische Codes gezeigt, das Gewicht der 4 Basiswörter, also derjenigen, die im Infoteil jeweils nur eine „1“ besitzen. Hier sind das v'(1), v'(2), v'(4) und v'(8), die alle die erforderlichen Gewichte von mindestens 7 aufweisen.

Paritätsgleichungen: Stars in der Manege

Nachdem nun bekannt ist, wie die Codewörter zu berechnen sind, muss noch geklärt werden, wie man zu den Paritätsgleichungen in Bild 3.17 kommt. In diesen liegt der eigentliche "Pfiff" des Goppa-Codes. Ein wesentlicher Teil der Aufgabe besteht in der Bestimmung der 2^m Polynome $h_i(z)$ vom Grad t-1. Goppa ging dabei von folgenden Überlegungen aus:

- Ein Polynom mit t Koeffizienten hat den Grad t-1. In unserem Fall ist t=3, ein geeignetes Polynom hat also den Grad 2.
- Ein Polynom (t-1)-ten Grades ist im GF(2^m) immer ein so genanntes modulares inverses Polynom zum linearen Polynom
  $(z - \beta_i)$ ersten Grades bezüglich eines Polynoms t-ten Grades. Die Bezeichnung β_i bedeutet hier willkürlich, aber ohne Einschränkung der Allgemeinheit:

  $$\beta_1 = 0,$$

  $$\beta_i = \beta^{i-2} \quad \text{für } i = 2, 3, 4, ..., 16$$

- Zur Kennzeichnung von n Codewortbits werden n verschiedene (= unabhängige) Polynome (t-1)-ten Grades benötigt, in unserem Fall also n=16.

Polynome – mal umgekehrt

- c(z) dient dazu, die bezüglich c(z) inversen Polynome $h_i(z)$ zu den linearen Polynomen $(z – \beta_i)$ zu berechnen. Wie ist das zu verstehen? Mathematisch betrachtet heißt es nichts anderes, als dass der Ausdruck

  $$h_i(z)\cdot(z - \beta_i) \text{ MOD } c(z) = 1$$

  gleich 1 ist. Von "normalen" Rechnungen her ist das bekannt: Wenn man eine Zahl mit ihrem inversen Part-

ner multipliziert, kommt immer das Ergebnis 1 heraus. In endlichen Körpern Z_p verhält es sich genauso, nur muss das Ergebnis der Multiplikation noch MOD p genommen werden. Im Z_5 gilt etwa: $3 \bullet 2$ MOD $5 = 1$, also ist 2 das inverse Element zu 3 (aber nur im Z_5 !), siehe auch Kapitel 3.2, Seite 44.

Bei Polynomen über endlichen Körpern verhält es sich nicht anders. Sehen wir uns das am Beispiel unseres Codes an und wählen als Polynom c(z)

$$c(z) = z^3 + z + 1,$$

welches in den 16 Elementen des GF(2^4) nach Tabelle 3-20 auf Seite145 tatsächlich keine Nullstellen hat (man prüfe das nach, indem man für z jedes der 16 „β_i" einsetzt).

Euklids Wunderwaffe

Zur Berechnung des modularen inversen Polynoms $h_i(z)$ zu $(z - \beta_i)$ bezüglich c(z) verwendet man den Euklidischen Algorithmus, siehe Kapitel 3.2, Seite 51 und bestimmt zunächst das größte gemeinsame Teilerpolynom. Das Ergebnis lässt sich hierfür bereits abschätzen, da die beiden Polynome so gewählt wurden, dass sie keine gemeinsamen Nullstellen haben, also *teilerfremd* sind. Dann muss ihr größtes gemeinsames Teilerpolynom ein Polynom nullten Grades, bzw. eine skalare Zahl sein. Mit der Festlegung

$$r_0 = c(z) = z^3 + z + 1,$$

$$r_1 = z - \beta_i$$

ergibt die unten als Nebenrechnung angegebene Polynomdivision ein Restpolynom vom Grad 0:

$$r_2(z) = 1 + \beta_i + \beta_i^3$$

Bewährte Rezepte

Der Euklidische Algorithmus wurde also im ersten Schritt bereits vollständig durchlaufen, so dass sich abkürzend schreiben lässt:

$$c(z) = q_1 \cdot (z - \beta_i) + r_2(z).$$

Nebenrechnung (Polynomdivision):

$$z^3 + z + 1 \quad : z - \beta_i \quad = z^2 + \beta_i z + (1 + \beta_i^2) + h(z)$$

$$\underline{\qquad\qquad\qquad}$$

$$z^3 + \beta_i z^2 + z$$

$$+\beta_i z^2 - \beta_i^2 z$$

$$\underline{\qquad\qquad\qquad}$$

$$(1+\beta_i^2)z + 1$$

$$(1+\beta_i^2)z + \beta_i(1+\beta_i^2)$$

$$\underline{\qquad\qquad\qquad}$$

$$+ 1+\beta_i+\beta_i^3 \rightarrow \text{Rest} = r_2(z),$$

$$\text{mit } h(z) = r_2(z)/(z - \beta_i)$$

Die Suche nach dem Inversen

Zur Bestimmung des gesuchten inversen Polynoms teilt man die Gleichung des ersten Schrittes zunächst durch den Rest $r_2(z)$:

$$c(z)\cdot r_2(z)^{-1} = (z^2 + \beta_i z + (1 + \beta_i^2))\cdot r_2^{-1}\cdot(z - \beta_i) + 1$$

und bildet links und rechts MOD c(z). Dabei erhält man

$$1 = 1 \text{ MOD } c(z)$$

$$= (z^2 + \beta_i z + (1 + \beta_i^2))\cdot r_2^{-1}\cdot(z - \beta_i) \text{ MOD } c(z)$$

wobei wegen der Charakteristik 2 im GF(2^4) die Vorzeichen nicht beachtet werden müssen. Man sieht, dass der Ausdruck

Wertvoller Fund!

$$(z^2 + \beta_i z + (1 + \beta_i^2))\cdot r_2^{-1} = h_i(z)$$

das gesuchte modulare inverse Polynom ist.

Erweitert man die linke Seite oben und unten mit $(z - \beta_i)$ und macht sich noch klar, dass der Rest r_2 nichts anderes als der Wert des Polynoms c(z) an der Stelle $z = \beta_i$ ist, so erhält man nach Ausmultiplikation von

$$(z^2 - \beta_i z + (1 - \beta_i^2))\cdot(z - \beta_i) = z^3 + z - \beta_i - \beta_i^3$$

$$= z^3 + z + 1 - r_2$$

die gleichwertige Darstellung

$$h_i(z) = \frac{c(z) - c(\beta_i)}{(z - \beta_i)}\cdot c(\beta_i)^{-1}, \quad \beta_1 = 0, \quad \beta_i = \beta^{i-2} \text{ für } i=2, 3, \ldots, n .$$

Man beachte, dass der Teilausdruck $(c(z) - c(\beta_i))$ immer eine Nullstelle in $z = \beta_i$ besitzt, da bei Einsetzen von $z = \beta_i$ der Wert 0 herauskommt. Er ist demnach durch $(z - \beta_i)$ ohne Rest teilbar, und man erhält aus ihm umgekehrt das Polynom $(z^2 + \beta_i z + (1+\beta_i^2))$ vom Grad (t-1) = 2.

pralle Fundgrube

Die Auswertung einer dieser beiden Darstellungen ergibt nun die 16 modularen inversen Polynome, welche in Tabelle 3-26 aufgelistet sind und bereits ausgiebig

zur Bildung des Codes benutzt wurden.

$h_1(z) = z^2 + 0z + 1$

$h_2(z) = z^2 + z + 0$

$h_3(z) = \beta^8 z^2 + \beta^9 z + \beta$

$h_4(z) = \beta z^2 + \beta^3 z + \beta^2$

$h_5(z) = \beta^{11} z^2 + \beta^{14} z + \beta^9$

$h_6(z) = \beta^2 z^2 + \beta^6 z + \beta^4$

$h_7(z) = \beta^{10} z^2 + z + 1$

$h_8(z) = \beta^7 z^2 + \beta^{13} z + \beta^3$

$h_9(z) = \beta^{10} z^2 + \beta^2 z + \beta^{13}$

$h_{10}(z) = \beta^4 z^2 + \beta^{12} z + \beta^8$

$h_{11}(z) = \beta^{13} z^2 + \beta^7 z + \beta^{12}$

$h_{12}(z) = \beta^5 z^2 + z + 1$

$h_{13}(z) = \beta^5 z^2 + \beta z + \beta^{14}$

$h_{14}(z) = \beta^{14} z^2 + \beta^{11} z + \beta^6$

$h_{15}(z) = \beta^{10} z^2 + \beta^8 z + \beta^7$

$h_{16}(z) = \beta^5 z^2 + \beta^4 z + \beta^{11}$

Tabelle 3-26: *modulare inverse Polynome im GF(2^4) zu ($z - \beta_i$) bezüglich des Polynoms $c(z) = z^3 + z + 1$*

Da man das modulare inverse Polynom $h_i(z)$ formal mit $(z - \beta_i)^{-1}$ bezeichnen kann, lässt sich die Paritätsbeziehung auch sehr kompakt mit der Formel in Bild 3.22a beschreiben.

$$\sum_{i=1}^{n} \frac{v_i}{(z-\beta_i)} \text{MOD}\, c(z) = 0, \quad \beta_1 = 0, \quad \beta_i = \beta^{i-2} \quad \text{für } i = 1, 2, \ldots\ldots, n$$

Bild 3.22a: *Kompaktdarstellung der Paritätsgleichung des Goppa-Codes*

Alles Gute ist einfach

Man sieht – vielleicht nach anfänglichen Zweifeln – dass die Grundgedanken, auf denen der Goppa-Code aufbaut, letztlich einfach sind. Zwei große Schönheitsfehler gibt es im Vergleich zu den bisher untersuchten Codierungsverfahren aber doch:

- Die Wahl des Polynoms c(z) ist innerhalb der gegebenen Grenzen (Teilerfremdheit zu $(z - \beta_i)$ und Grad c(z)

= t-1), willkürlich. Das heißt, dass mit größerem Umfang des Erweiterungskörpers GF(2^m) und wachsender Anzahl zu korrigierender Fehler auch die Zahl wählbarer Polynome c(z) wächst, ohne dass ein technisch eindeutiges Kriterium vorhanden wäre, um *ein* ganz bestimmtes nehmen zu müssen.

BCH: Sieger nach Punkten

- die Informationsrate R ist für kleine zu korrigierende Fehlerzahlen (< 100) geringer als etwa beim BCH-Code. Unser Beispiel aus Unterkapitel 3.7.7 bot für 3-Fehler-Korrektur bei einer Länge n=15 Platz für 5 Informationsstellen. Das entspricht einer Informationsrate von R = 33,3%, während wir bei unserem Goppa-Code nur über 25% verfügen. Der vergleichbare BCH-Code transportiert also mit der gleichen Anzahl von Bits immerhin die Hälfte mehr an Informationen!

Goppa-Special: Geheimhaltung

- Daher lohnt sich der Einsatz des Goppa-Codes eigentlich nicht, wenn man ausschließlich nur Fehler korrigieren will. Anders sieht es dagegen aus, wenn man ihn zum Verschlüsseln von Klartexten in Geheimtexte benutzt, siehe Unterkapitel 5.7. Hier stellt die willkürliche Wahlmöglichkeit für c(z) einen Zusatzvorteil für die "Verschleierung" dar, den man wie einen Geheimschlüssel gebraucht. Die weniger gute Informationsrate stört in diesem Zusammenhang ohnehin nicht so sehr.

3.8.2 Zwei Lösungswege für die Decodierung

Nun folgt ein Blick auf das Decodieren des Goppa-Codes. Es gibt hierfür zwei verschiedene Lösungswege, von denen der zuerst behandelte (hoffentlich) ein einfacheres Verständnis gestattet, während der zweite sich besser für die Abarbeitung mit Rechenprogrammen eignet, weil hier der

Großteil der erforderlichen Operationen numerisch gelöst werden kann.

Zwei Wege führen nach Goppa

Zunächst betrachten wir noch die für beide Lösungen gemeinsam zugrunde zu legenden Ausgangsbeziehungen. Wie bisher üblich bezeichnet man den mit einem Fehlervektor e verfälschten Codewortvektor v als Empfangswortvektor w:

$$w = v + e$$

oder in Polynom-Schreibweise:

$$w(z) = \sum_{i=1}^{n} \frac{w_i}{z-\beta_i} \text{MOD } c(z) = \sum_{i=1}^{n} \left(\frac{v_i}{z-\beta_i} + \frac{e_i}{z-\beta_i} \right) \text{MOD } c(z)$$

$$= \sum_{i=1}^{n} \frac{e_i}{z-\beta_i} \text{MOD } c(z) \quad = \quad s(z)$$

Eine Verfälschung erkennt man daran, dass diese Paritätsgleichung nicht mehr Null ist, was sie bei Fehlerfreiheit aber sein müsste. Da die Paritätsgleichung ein Polynom in z vom Grad (t-1)=2 darstellt, erhält man daraus nach Einsetzen der bekannten Empfangswortelemente w_i (statt der unbekannten Codewortelemente v_i) ein Syndrompolynom

$$s(z) = s_2 z^2 + s_1 z + s_0$$

Wo liegt der Fehler?

Mit dessen Hilfe lassen sich die unbekannten Fehlerpositionen bestimmen. Dazu definiert man das Fehlerpositions-Polynom

$$\sigma(z) = (z - \beta_j) \bullet (z - \beta_k) \bullet (z - \beta_l) = z^3 + \sigma_2 z^2 + \sigma_1 z + \sigma_0$$

in welchem die Indices j, k und l die Fehlerpositionsnummern und β_j, β_k und β_l die zugehörigen Potenzen des primitiven Elements sind, also $\beta j = \beta^j$ usw.

Partnervermittlung für das Syndrompolynom

Das Syndrompolynom s(z) kennt man nur in der oben dargestellten Form $s(z) = s_2 z^2 + s_1 z + s_0$, da sich die Koeffizienten aus den empfangenen Elementen w_i berechnen lassen. Die zuvor aufgestellte Summenschreibweise aus den Fehleranteilen ist dagegen in dieser Form nicht zugänglich! Allerdings erhält man ein sehr nützliches Ergebnis, wenn man das Syndrompolynom s(z) mit dem Fehlerpo-

sitions-Polynom σ(z) multipliziert:

$$s(z)\cdot\sigma(z)=\left(\sum_{i=j,k,l}\frac{e_i}{z-\beta_i}\right)\cdot\left((z-\beta_j)\cdot(z-\beta_k)(z-\beta_l)\right)$$

$$=(z-\beta_k)\cdot(z-\beta_l)+(z-\beta_j)\cdot(z-\beta_l)+(z-\beta_j)\cdot(z-\beta_k)$$

$$=\frac{d\sigma(z)}{dz}\qquad=\sigma'$$

Ableiten ohne Differenzieren

Das Produkt ergibt also nichts anderes als die Ableitung des Fehlerpositions-Polynoms nach dz.

Hinweis: die Ableitung eines Produktausdruckes

$y(z) = (z-a)\cdot(z-b)\cdot(z-c)$

nach dz erfolgt nach der Produktregel

$y'(z) = (z-b)\cdot(z-c) + (z-a)\cdot(z-c) + (z-a)\cdot(z-b).$

Da man einerseits den Wert dieses Ausdrucks aus den zugänglichen Empfangswortelementen w_i berechnen kann, nämlich

$$s(z)\cdot\sigma(z) = \left(\left(\sum_{i=1}^{n}\frac{w_i}{z-\beta_i}\right)\cdot(z^3+\sigma_2 z^2+\sigma_1 z+\sigma_0)\right)\text{MOD } c(z)$$

andererseits aber auch die Ableitung σ'(z) kennt, also

$$\sigma'(z) = 3z^2 + 2\sigma_2 z + \sigma_1 = z^2 + \sigma_1$$

lässt sich hieraus eine Beziehung aufstellen, mit der die Fehlerpositionswerte β_i berechenbar werden. Man beachte dabei, dass bei der Bildung der Ableitung des Fehlerpositions-Polynoms wegen der Charakteristik p=2 alle Vielfachen von 2 verschwinden. Es ist:

$$s(z)\cdot\sigma(z) = \left(\left(\sum_{i=1}^{n}\frac{w_i}{z-\beta_i}\right)\cdot(z^3+\sigma_2 z^2+\sigma_1 z+\sigma_0)\right)\text{MOD } c(z)$$

$$= z^2+\sigma_1$$

Das Syndrompolynom s(z) liegt nach Auswertung der „w_i's" auch in der Form $s(z) = s_2z^2 + s_1z + s_0$ vor, wobei die Koeffizienten s_0, s_1 und s_2 ganz bestimmte Werte anneh-

men, welche von den aufgetretenen Fehlern abhängen. Daher lässt sich schreiben:

$$s(z)\cdot\sigma(z)=\Big((s_2z^2+s_1z+s_0)\cdot(z^3+\sigma_2z^2+\sigma_1z+\sigma_0)\Big)\,\mathrm{MOD}\,c(z)$$
$$=z^2+\sigma_1$$

Die großen Unbekannten

Hierin sind nur die drei Koeffizienten des Fehlerpositions-Polynoms σ(z) unbekannt.

Beim *ersten* Lösungsweg benutzt man einen Koeffizientenvergleich der in dieser Beziehung enthaltenen Polynome, um daraus ein lineares Gleichungssystem zu ihrer Bestimmung zu gewinnen.

Zunächst multipliziert man das Syndrom-Polynom mit dem Fehlerpositions-Polynom:

$$(s_2z^2+s_1z+s_0)\cdot(z^3+\sigma_2z^2+\sigma_1z+\sigma_0)$$
$$=s_2z^5+(s_1+s_2\sigma_2)z^4+(s_0+s_1\sigma_2+s_2\sigma_1)z^3$$
$$+(s_0\sigma_2+s_1\sigma_1+s_2\sigma_0)z^2+(s_0\sigma_1+s_1\sigma_0)z+s_0\sigma_0\,.$$

Divisionen marsch ins Gefecht!

Auf dieses Ergebnis wendet man die MOD c(z)- Funktion an, d. h., man sucht den Rest bezüglich $c(z)=z^3+z+1$. Die dazu auszuführende Polynomdivision (negative Vorzeichen haben, wie immer im GF(2^m), die gleiche Bedeutung wie positive, und man kann sich daher auf letztere beschränken) führt auf

$$s(z)\cdot\sigma(z):c(z) = s_2z^2+(s_1+s_2\sigma_2)z+(s_0+s_1\sigma_2+s_2\sigma_1+s_2)$$
$$+\,\mathrm{Rest}/c(z),$$

wobei der Rest ein Polynom 2.Grades in z darstellt:

$$\mathrm{Rest} = [(s_0+s_2)\sigma_2+s_1\sigma_1+s_2\sigma_0+s_1+s_2]z^2$$
$$+[(s_1+s_2)\sigma_2+(s_0+s_2)\sigma_1+s_1\sigma_0+s_0+s_1+s_2]z$$
$$+[s_1\sigma_2+s_2\sigma_1+s_0\sigma_0+s_0+s_2].$$

Tolle Vergleiche

Bei einem Koeffizientenvergleich mit dem abgeleiteten Fehlerpositions-Polynom $\sigma'(z)=z^2+\sigma_1$ entsteht schließlich ein lineares Gleichungssystem 3.Ordnung in den Unbekannten σ_2, σ_1 und σ_0:

$$(s_0+s_2)\sigma_2+s_1\sigma_1+s_2\sigma_0 = 1+s_1+s_2$$
$$(s_1+s_2)\sigma_2+(s_0+s_2)\sigma_1+s_1\sigma_0 = s_0+s_1+s_2$$

$$s_1\sigma_2 \quad + (1+s_2)\,\sigma_1 \quad + s_0\sigma_0 \quad = s_0 + s_2.$$

Ein Beispiel sagt mehr als alle Theorie

Wir überprüfen dieses Ergebnis mit Hilfe eines Beispiels und nehmen dazu an, dass gemäß Bild 3.21 das Codewort v'(3) gesendet wurde, welches aber mit einem 3-Bit-Fehler e verfälscht ist, so dass der Empfänger das Wort

$$w = v'(3) + e$$

erhält. Das Empfangswort w sieht dann folgendermaßen aus:

$$v'(3) = [1\,1\,1\,1 \;\; 0\,1\,0\,0 \;\; 1\,1\,0\,1 \;\; 0\,0\,1\,1]$$
$$e = [0\,1\,1\,1 \;\; 0\,0\,0\,0 \;\; 0\,0\,0\,0 \;\; 0\,0\,0\,0]$$
$$\_\_\_\_\_$$
$$w = [1\,0\,0\,0 \;\; 0\,1\,0\,0 \;\; 1\,1\,0\,1 \;\; 0\,0\,1\,1].$$

Dieses Empfangswort wird von rechts an die Paritätsprüfmatrix $H^{\bullet}$ aus Bild 3.22 heran multipliziert, so dass das Syndromwort

$$s = [0\,1\,1\,0 \;\; 0\,0\,1\,1 \;\; 0\,1\,1\,0]$$

entsteht, welches nach Zusammenfassung von je 4 Stellen gleichwertig auch als Syndrompolynom 2. Grades aufgefaßt werden kann. Da das Tupel 0110 die Polynom-Darstellung von β^5 und das Tupel 0011 die von β^4 ist (siehe Tabelle 3-20 auf Seite 145), haben wir als Syndrompolynom also

$$s(z) = s_2 z^2 + s_1 z + s_0 = \beta^5 z^2 + \beta^4 z + \beta^5.$$

Gleichungen mit System

Damit nimmt das Gleichungssystem die Form

$$\begin{bmatrix} 0 & \beta^4 & \beta^5 \\ \beta^8 & 0 & \beta^4 \\ \beta^4 & \beta^{10} & \beta^5 \end{bmatrix} \cdot \begin{bmatrix} \sigma_2 \\ \sigma_1 \\ \sigma_0 \end{bmatrix} = \begin{bmatrix} \beta^2 \\ \beta^4 \\ 0 \end{bmatrix}$$

an. Die Lösung erfolgt wieder so, wie sie schon im Unterkapitel 3.7.7 für den BCH-Code beschrieben wurde. Die drei unbekannten Koeffizienten ermittelt man dabei als

$$\sigma_2 = \beta^{10}$$
$$\sigma_1 = \beta^{11}$$
$$\sigma_0 = \beta^3.$$

Damit liegt auch das Fehlerpositions-Polynom

$$\sigma(z) = z^3 + \beta^{10} z^2 + \beta^{11} z + \beta^3$$

vor, und man kann daraus die Nullstellen β_j, β_k und β_l bestimmen. Das geschieht am einfachsten durch systematisches Durchprobieren aller Elemente des GF(2^4). Ergebnis: $\sigma(z) = 0$ für

$$z_1 = \beta^0$$

$$z_2 = \beta^1$$

$$z_3 = \beta^2.$$

Interpreten gesucht ...

Wie ist dieses Ergebnis zu interpretieren? Jede Nullstelle ist derjenigen Code-Elementposition zugeordnet, für die mit ihr das entsprechende modulare inverse Polynom berechnet wurde, siehe Bild 3.22a, siehe Seite 195. Das hört sich komplizierter an, als es ist. Der Wert "0" steht danach für die 1. Position, β^0 für die 2. Position, β^1 für die 3.Position, β^2 für die 4. Position usw. Also sind im Beispiel die Fehler an der 2., 3. und 4. Stelle im empfangenen Wort w aufgetreten und können korrigiert werden.

Erinnerung an vertauschte Rollen

Aufpassen muss man mit den vertauschten Stellen 12 und 13. Durch die Vertauschung ist β^{10} für die Stelle 13 und β^{11} für die Stelle 12 "zuständig". Ein Beispiel: Angenommen, der Fehler kippe wieder 3 Bits, die folgendermaßen verteilt sind:

$$e = [0\,0\,0\,0 \;\; 0\,1\,0\,1 \;\; 0\,0\,0\,1 \;\; 0\,0\,0\,0].$$

Das Syndrom liefert

$$s = [1\,0\,0\,1 \;\; 0\,0\,1\,1 \;\; 0\,0\,1\,0]$$

und entspricht damit dem Syndrompolynom

$$s(z) = \beta^{14} z^2 + \beta^4 z + \beta^1.$$

Die Auswertung auf dem beschriebenen Weg gibt das Fehlerpositions-Polynom

$$\sigma(z) = z^3 + z^2 + \beta^1 z + \beta^6,$$

wozu die drei Nullstellen

$$z = \beta^4$$

$$z = \beta^6$$

$$z = \beta^{11}$$

gehören. Diese wiederum entsprechen den Fehlerpositionen 6, 8 und 12 (**nicht 13!**).

Die somit erarbeitete Lösung eignet sich zunächst nur für den Fall, dass genau 3 Fehler aufgetreten sind. Was aber passiert, wenn es nur 2 oder 1 Fehler sind? Dann lässt sich das Gleichungssystem, welches ja für den Fall von 3 Fehlern aufgestellt wurde, nicht mehr lösen, da weniger als 3 unabhängige Gleichungen vorliegen. Man erkennt das daran, dass die Systemdeterminante den Wert 0 annimmt. Dann müssen die abhängigen Gleichungen entfernt werden, oder man schaltet auf einen bereits vorbereiteten Lösungsalgorithmus für 2 Fehler um usw.

... damit die Fälle nicht wegschwimmen

Man kann alles weiter vereinfachen, wenn man sich eine allgemeine Lösung für lineare Gleichungssysteme beliebiger Ordnung im GF(2^m) aufbaut, was nach dem bisher Erarbeiteten nicht schwer fallen dürfte. Lästig bleibt bei diesem Vorgehen allerdings immer die Notwendigkeit, die Bildungsvorschriften für die System-Koeffizienten für jede Maximalzahl zu korrigierender Fehler und für jedes Polynom c(z) neu aufstellen zu müssen. Wenn dies auch stets ohne weiteres möglich ist, lohnt es sich doch, nach weiteren Verfahren zu suchen, die zur Entlastung beitragen.

Es geht auch anders

Der nun betrachtete *zweite Lösungsweg* ist hierfür ein interessanter, aussichtsreicher Kandidat und wurde von *N.J. Patterson* vorgeschlagen ("The Algebraic Decoding of Goppa Codes", IEEE Transactions on Information Theory, Volume IT-21, Nr.2, Kapitel V, Seite 207, März 1975). Das Verfahren geht von der bereits benutzten Beziehung

$$s(z)\cdot\sigma(z) \text{ MOD } c(z) = \sigma'(z)$$

Leckerbissen für unsere digitalen Freunde

aus und löst diese mit Hilfe des Euklidischen Algorithmus direkt, s.S. 51. Der Vorteil gegenüber dem ersten Lösungsweg liegt vor allem darin, dass er nicht so vieler Vorbereitungen bedarf und sich statt dessen in größerem Umfang numerisch – und damit auf einem Rechner – bearbeiten lässt.

Patterson stellt zunächst fest, dass sich das Fehlerpositions-Polynom in der Form

$$\begin{aligned}\sigma(z) &= (\sigma_2 z^2 + \sigma_0) &&+ z\cdot(\sigma_3 z^2 + \sigma_1)\\ &= \delta^2(z) &&+ z\cdot\epsilon^2(z)\end{aligned}$$

schreiben lässt, wobei der Koeffizient σ_3 im allgemeinen 1 ist. Beide Teilpolynome $\delta^2(z)$ und $\epsilon^2(z)$ enthalten nur geradzahlige Potenzen von z:

$$\delta^2(z) = \delta_1^2 z^2 + \delta_0^2$$

$$\varepsilon^2(z) = \varepsilon_1^2 z^2 + \varepsilon_0^2.$$

Bei Bildung der formalen Ableitung nach z verschwinden diese Teilpolynome, da die geradzahligen Exponenten als Faktoren vor die Summanden wandern und bei MOD 2-Ausführung den Wert 0 annehmen. Deshalb ist

$$\sigma'(z) = \delta^{2\prime} + z' \cdot \varepsilon^2(z) + z \cdot \varepsilon^{2\prime}(z) = \varepsilon^2(z).$$

Damit kann die obige Ausgangsbeziehung auch in der Form

$$s(z) \cdot \sigma(z) \text{ MOD } c(z) = \sigma'(z)$$
$$= s(z) \cdot (\delta^2(z) + z \cdot \varepsilon^2(z)) \text{ MOD } c(z) = \varepsilon^2(z)$$

Der EA und seine Inversen

geschrieben werden. An dieser Stelle sei daran erinnert, dass sich der Euklidische Algorithmus dann mit Vorteil einsetzen lässt, wenn man zu teilerfremden Polynomen das modulare inverse Polynom berechnen will. Dieses Prinzip lässt sich noch weiter "ausschlachten", wenn man nicht f(z) in

$$s(z) \cdot f(z) \text{ MOD } c(z) = 1$$

sondern σ(z) in

$$s(z) \cdot \sigma(z) \text{ MOD } c(z) = \varepsilon^2(z)$$

sucht. Man läuft dann den Euklidischen Algorithmus nicht bis zu demjenigen Schritt durch, bei dem der Rest ein Polynom nullten Grades (eine "Zahl") ist, sondern beendet ihn bereits, wenn als Rest ein Polynom des Grades von $\epsilon^2(z)$ auftritt. Da die Lösung für einen vorgegebenen Polynomgrad eindeutig ist, wäre das dann die gesuchte. Man muss also

$$c(z) = q_1(z) \cdot s(z) + r_2(z)$$

berechnen, was numerisch ohne weiteres gelingt, da sowohl c(z) als auch s(z) vollständig bekannt sind. Eine kurze Überprüfung zeigt uns allerdings, dass wir mit dieser Ausgangsform unmittelbar noch nicht weiterkommen, da hier der Grad des Restpolynoms $r_2(z)$ wegen Grad s(z) = 2 nur kleiner als 2 sein kann, was im Widerspruch zur allgemeinen Form von $\epsilon_2(z)$ steht. *Patterson* macht daher zunächst eine passende Umformung und verwendet in dieser das modulare inverse Polynom f(z) zu s(z):

$$s(z) \cdot f(z) \text{ MOD } c(z) = 1.$$

Muss sein: Saubere Vorbereitung

Man sieht, dass f(z) rein numerisch ermittelt werden kann, da das Syndrompolynom s(z) bekannt ist. Nun multipliziert man f(z) an

$$f(z)\cdot s(z)\cdot\sigma(z) \text{ MOD } c(z) = f(z)\cdot s(z)\cdot(\delta^2(z) + z\cdot\varepsilon^2(z)) \text{ MOD } c(z)$$
$$= 1\cdot(\delta^2(z) + z\cdot\varepsilon^2(z)) \text{ MOD } c(z)$$
$$= f(z)\cdot\varepsilon^2(z)$$

heran und erhält nach Umstellung

$$(f(z) + z)\cdot\varepsilon^2(z) \text{ MOD } c(z) = \delta^2(z).$$

Nach den Rechenregeln für die MOD-Funktion ist

$$(f(z) + z)\cdot\varepsilon^2(z) \text{ MOD } c(z) = ((f(z) + z) \text{ MOD } c(z))$$
$$\cdot(\epsilon^2(z) \text{ MOD } c(z)) \text{ MOD } c(z)$$
$$= \delta^2(z),$$

wobei sich für (f(z) + z) MOD c(z) auch das Quadrat eines Polynoms κ(z) einsetzen lässt (siehe *Patterson*, S. 207). *Hinweis:* Das Polynom (f(z) + z) enthält in seinen Koeffizienten zwar alle Informationen zur Fehlerposition, es ist im allgemeinen aber weder von seinem Aufbau her noch nach seinem Grad zur Weiterverwendung in dem eingeschlagenen Lösungsweg geeignet. Die nachstehend beschriebene Umformung ergibt eine passende Darstellung bei vollständigem Erhalt aller Informationen:

$$(f(z) + z) \text{ MOD } c(z) = \kappa^2(z) \text{ MOD } c(z).$$

Uff, ganz schön viel, aber bald gibt es eine satte ..

Dieses Hilfspolynom κ(z) kann immer *eindeutig* durch einen entsprechenden Ansatz

$$\kappa(z) = \kappa_2 z^2 + \kappa_1 z + \kappa_0$$

gefunden werden. Die Quadrate κ_2^2, κ_1^2 und κ_0^2 der Koeffizienten ergeben sich als lineare Funktionen aus der obigen MOD c(z)-Beziehung. Mit

$$\kappa^2(z) = \kappa_2^2 z^4 + \kappa_1^2 z^2 + \kappa_0^2$$

kann diese als

$$(f(z) + z)\cdot\varepsilon^2(z) \text{ MOD } c(z) = \kappa^2(z)\cdot\varepsilon^2(z) \text{ MOD } c(z) = \delta^2(z)$$

geschrieben werden, und unsere Aufgabe hat sich in die Lösung von

$$\kappa(z)\cdot\varepsilon(z) \text{ MOD } c(z) = \delta(z)$$

verwandelt, da diese Form wegen

$$(\kappa(z)\cdot\varepsilon(z) \text{ MOD } c(z))^2 = (\kappa(z)\cdot\varepsilon(z))^2 \text{ MOD } c(z)$$

$$= (\kappa^2(z)\cdot\varepsilon^2(z))^2 \text{ MOD } c(z) \quad = \delta^2(z)$$

... Aufwandsentschädigung

mit ihrer quadratischen "Schwester" gleichbedeutend ist. Nun lässt man den Euklidischen Algorithmus ablaufen und sucht denjenigen Schritt, bei dem sich als Restpolynom eines vom Grad δ(z) ergibt. Dann hat man ein Polynompaar gefunden, welches – in eindeutiger Weise – die Beziehung

$$\kappa(z)\cdot\varepsilon(z) \text{ MOD } c(z) \quad = \delta(z)$$

erfüllt. Damit liegt auch das Fehlerpositions-Polynom σ(z) vor

Hinweis: Das Rechnen mit Quadraten ist in Galoisfeldern $GF(2^m)$ deshalb vorteilhaft, weil es sowohl zu jedem Element eindeutig ein quadratisches Element gibt, als auch umgekehrt zu jedem Element eindeutig eine Wurzel existiert. Das wird hier ausgenutzt. Beispiele: $(\beta^8)^2 = \beta^{16} = \beta^1$, $(\beta^3)^{-2} = (\beta^{18})^{-2} = \beta^9$.

Führen wir ein Beispiel aus und verwenden wieder unser Syndrompolynom für die bereits gemachte Annahme dreier Fehler. Es ist in diesem Fall

$$s(z) \quad = \beta^5 z^2 + \beta^4 z + \beta^5.$$

1. Schritt

Ein Beispiel, bitte!

Das modulare inverse Polynom f(z) berechnet sich aus c(z) und s(z) als

$$c(z) \quad = q_1(z)\cdot s(z) + r_2(z)$$

$$s(z) \quad = q_2(z)\cdot r_2(z) + r_3(z)$$

usw., wobei

$$c(z) \quad = (\beta^{10}z+\beta^9)\cdot s(z) \quad + (\beta^{13}z+\beta^3)$$

$$s(z) \quad = (\beta^7 z+\beta^4)\cdot(\beta^{13}z+\beta^3) \quad + \beta^{13}$$

ist. Die zweite Gleichung löst man nach β^{13} auf und multipliziert beide Seiten mit $\beta^{-13} = \beta^2$:

$$1 \quad = \beta^{-13}\cdot s(z) - \beta^{-13}\cdot(\beta^7 z+\beta^4)\cdot(\beta^{13}z+\beta^3).$$

Die erste Gleichung liefert

$$r_2(z) \quad = (\beta^{13}z+\beta^3)$$

$$= c(z) \quad - (\beta^{10}z+\beta^9)\cdot s(z)$$

und dieses Ergebnis wird in die zweite Gleichung eingesetzt:

$$1 = \beta^{-13}\cdot s(z) - \beta^{-13}\cdot(\beta^7 z+\beta^4)\cdot(c(z) - (\beta^{10} z+\beta^9)\cdot s(z)).$$

Eine kleine Umformung ergibt

$$1 = \beta^{-13}\cdot(1 +(\beta^7 z+\beta^4)\cdot(\beta^{10} z+\beta^9))\cdot s(z) - \beta^{-13}\cdot(\beta^7 z+\beta^4)\cdot c(z)$$

und die MOD c(z)- Bildung auf beiden Seiten zeigt, dass mit

$$f(z) = \beta^{-13}\cdot(1 +(\beta^7 z+\beta^4)\cdot(\beta^{10} z+\beta^9)) = \beta^4 z^2 + \beta^9 z + \beta^8$$

Bestätigung

tatsächlich das modulare inverse Polynom zu s(z) vorliegt.

2. Schritt

Wir berechnen ein Polynom κ(z) so, dass

$$\kappa^2(z) \text{ MOD } c(z) = f(z) + z = \beta^4 z^2 + \beta^7 z + \beta^8$$

herauskommt. (*Hinweis*: Auf die rechte Seite f(z) + z muss in diesem Fall die MOD c(z)-Funktion nicht angewendet werden, da f(z) + z bereits einen kleineren Grad als c(z) hat). Versuchen wir dazu den Ansatz

$$\kappa(z) = \kappa_2 z^2 + \kappa_1 z + \kappa_0 ,$$

was beim Quadrieren das Polynom

$$\kappa^2(z) = \kappa_2^2 z^4 + \kappa_1^2 z + \kappa_0^2$$

ergibt. Nach MOD c(z)-Bildung erhält man

$$(\kappa_1^2+\kappa_2^2) z^2 + \kappa_2^2 z + \kappa_0^2 = \beta^4 z^2 + \beta^7 z + \beta^8.$$

Ein Koeffizientenvergleich liefert

$$\kappa_0^2 = \beta^8 \rightarrow \kappa_0 = \beta^4$$

$$\kappa_2^2 = \beta^7 \rightarrow \kappa_2 = \beta^{11}$$

$$\kappa_1^2 = \kappa_2^2 + \beta^4 = \beta^3 \rightarrow \kappa_1 = \beta^9 .$$

Das ist übrigens die einzige Stelle in diesem Algorithmus, an der "analytisch" gearbeitet werden muss, d. h. wo eine formale Rechnung auszuführen ist. Diese verlangt allerdings wesentlich weniger Aufwand als diejenige im ersten Lösungsweg.

Schritt für Schritt

3. Schritt

Nun kann man den oben erwähnten Ausdruck

$$\kappa(z)\cdot\varepsilon(z) \text{ MOD } c(z) = \delta(z)$$

mit Hilfe des Euklidischen Algorithmus lösen. Dazu setzt man

$$c(z) = q_1(z)\cdot\kappa(z) + r_2(z)$$

an und erhält mit

$$\kappa(z) = \beta^{11} z^2 + \beta^9 z + \beta^4$$

$$z^3 + z + 1 = (\beta^4 z+\beta^2)\cdot(\beta^{11} z^2+\beta^9 z+\beta^4) + (\beta^9 z+\beta^{13})$$

Der Charme des (Dienst-) Grades

im ersten Schritt als Rest bereits ein Polynom vom Grad

$$\delta(z) = \delta_1 z + \delta_0,$$

so dass die Lösung

$$r_2(z) \text{ MOD } c(z) = c(z) \text{ MOD } c(z) + q_1(z)\cdot\kappa(z) \text{ MOD } c(z)$$

$$= q_1(z)\cdot\kappa(z) \text{ MOD } c(z)$$

die gesuchte sein wird. Damit stehen aber auch die beiden Teilpolynome

$$\delta(z) = \beta^9 z + \beta^{13}$$

$$\varepsilon(z) = \beta^4 z + \beta^2$$

und ihre Quadrate fest, so dass nun das Fehlerpositions-Polynom

$$\sigma(z) = \beta^8 z^3 + \beta^3 z^2 + \beta^4 z + \beta^{11}$$

entsteht. Da die Nullstellen eines Polynoms unabhängig von einem gemeinsamen Faktor in den Koeffizienten sind, können wir es noch mit $\beta^{-8} = \beta^7$ multiplizieren und erhalten das Ergebnis

$$\sigma_a(z) = z^3 + \beta^{10} z^2 + \beta^{11} z + \beta^3,$$

was genau dem auf dem ersten Lösungsweg ermittelten entspricht. Über die Nullstellen im GF(2^4) gibt es ebenfalls die Fehlerpositionen 2, 3 und 4 richtig an.

Was passiert, wenn *weniger als 3 Fehler* aufgetreten sind, was aus dem Syndrompolynom nicht zu erkennen ist? Sehen wir uns zunächst den 1-Bit-Fehlerfall an, der das Codewort mit dem Fehler

$$e = [0\,0\,1\,0 \;\; 0\,0\,0\,0 \;\; 0\,0\,0\,0 \;\; 0\,0\,0\,0]$$

Geht's auch mit etwas weniger?

verfälscht. Das Syndrompolynom ist dann

$$s(z) = \beta^8 z^2 + \beta^9 z + \beta,$$

wozu das inverse Polynom f(z)

$$f(z) = z + \beta$$

gehört. Damit kann man im 2. Lösungsschritt aufsetzen und berechnet die Koeffizienten des Polynoms κ(z) als

$\kappa_0^2 = \beta$ $\kappa_0 = \beta^8$

$\kappa_2^2 = 0$ $\kappa_2 = 0$

$\kappa_1^2 = \kappa_2^2 + 0 = 0$ $\kappa_1 = 0.$

Der Hilfssheriff und seine Grenze

Das Hilfspolynom $\kappa(z) = \beta^8$ "entartet" hier also zu einem solchen nullten Grades. Wenn man hiermit den Euklidischen Algorithmus ablaufen lässt, kommt man scheinbar nicht weiter, da

$$c(z) = (c(z)\cdot\beta^{-8})\cdot\kappa(z) + 0$$

entsteht. Damit gibt aber der Euklidische Algorithmus indirekt einen Hinweis, dass die Lösung von

$$\kappa(z)\cdot\varepsilon(z) \text{ MOD } c(z) = \delta(z)$$

ohne Bildung der MOD c(z)-Funktion zu finden ist, da in dem Ausdruck links gar keine Potenzen von z größer als 2 auftreten. Anders ausgedrückt heißt dies, dass man einfach einen unmittelbaren Koeffizientenvergleich der links und rechts stehenden Polynome durchzuführen hat:

$$\beta^8\cdot(\epsilon_1 z + \varepsilon_0) = \delta_1 z + \delta_0.$$

Da für die 4 unbekannten Koeffizienten nur 2 Bestimmungsgleichungen vorhanden sind, setzt man einen Koeffizienten willkürlich auf 0 und wählt dazu denjenigen, welcher im Fehlerpositions-Polynom $\sigma(z)$ bei der höchsten z-Potenz steht, nämlich ϵ_1. Damit wird aber automatisch auch $\delta_1 = 0$ und es bleibt

$$\beta^8\cdot\varepsilon_0 = \delta_0.$$

Das Fehlerpositions-Polynom sieht nun so aus

$$\sigma(z) = \varepsilon_0^2 z + \delta_0^2,$$

... und klappt ...

und man erkennt an seinem Grad, dass nur ein 1-Bit-Fehler aufgetreten war. Da man zur Nullstellenbestimmung einen gemeinsamen Faktor herausziehen darf, ohne an der Nullstelle etwas zu verändern, kann δ_0 willkürlich auf den Wert 1 gesetzt werden, und für den anderen Koeffizienten ergibt sich

$$\varepsilon_0^2 = (1\cdot\beta^{-8})^2 = \beta^{14}.$$

Die Nullstelle hat also den Wert

$$z_0 = \beta^1,$$

was tatsächlich der 3. Position im Codewort entspricht.

Für den Fall *zweier* 1-Bit-Fehler versuche man einmal, selbst die Lösung zu finden und gehe dazu z. B. vom Fehler

$e = [0\,0\,1\,0\;\;0\,0\,0\,0\;\;1\,0\,0\,0\;\;0\,0\,0\,0]$,

aus, der das Syndrompolynom

$s(z) = \beta^{1} z^{2} + \beta^{11} z + \beta^{12}$

liefert.

Kontrollinformationen und Hinweise:

- $f(z) + z = \beta^{1} z^{2} + \beta^{9} - \kappa_2 = 0$,
- $\kappa_1 = \beta^{8}$, $\kappa_0 = \beta^{12}$,
- Man führe einen direkten Koeffizientenvergleich ohne MOD c(z)-Bildung durch (wie zuvor), da der Euklidische Algorithmus nicht auf eine verträgliche Lösung führt und damit ein "Signal" gibt.
- Man setze ϵ_1 auf 0. Ergebnisse:
  $\delta_1 = 1$ (willkürlich), $\delta_0 = \beta^{4}$, $\varepsilon_0 = \beta^{7}$.

Grade, gerade und ungerade

Allgemein gibt es Aussagen zum Grad, welchen die Polynome $\delta^2(z)$ und $\epsilon^2(z)$ gegenseitig aufweisen müssen, damit der Euklidische Algorithmus im 3. Lösungsschritt (nicht zu verwechseln mit einem 3. Schritt innerhalb des Euklidischen Algorithmus) ablaufen kann. Diese Aussagen lassen sich aus folgenden Überlegungen ableiten:

- Ist die *maximale* Anzahl zu korrigierender Fehler *ungerade* (in unserem Beispiel ist n=3), dann stimmen die Polynomgrade überein, im Beispiel sind beide Polynome vom Grad 2.
- Ist die *maximale* Anzahl zu korrigierender Fehler aber *gerade*, dann hat $\delta^2(z)$ einen um 2 höheren Grad als $\epsilon^2(z)$, oder – anders ausgedrückt – , der Grad von $\delta(z)$ ist um 1 höher als der von $\epsilon(z)$.

Lässt man für

$\kappa(z)\cdot\varepsilon(z)\ \mathrm{MOD}\ c(z) = \delta(z)$

den Euklidischen Algorithmus ablaufen, so werden die Polynome $q_1(z)$ und $r_2(z)$ in der Beziehung

$c(z) = q_1(z)\cdot\kappa(z) + r_2(z)$

diese Bedingungen einhalten, wenn sie Lösungskandidaten sind. Andernfalls signalisiert uns das Ergebnis, dass wir die MOD c(z)- Funktion nicht ausführen müssen, wie wir es im Beispiel zuvor gesehen hatten. So etwa erhält man für den genannten 2-Bit-Fehlerfall im ersten Schritt des Euklidischen Algorithmus mit

$$c(z) = (\beta^7 z^2+\beta^{11} z+\beta^9)\cdot\kappa(z) + \beta^{13},$$

ein Restpolynom $r_2(z)$ vom Grad 0 und ein Polynom $q_1(z)$ vom Grad 2, was zu einem direkten Koeffizientenvergleich veranlaßt.

Damit sind zwei unterschiedliche Lösungswege bekannt, und wir schließen die Betrachtung des Goppa-Codes mit folgenden Bemerkungen ab:

BCH-Code ist Spezialfall

- Der BCH-Code entsteht als Sonderfall aus dem Goppa-Code, wenn man als erzeugendes Polynom c(z) einfach

  $$c(z) = z^{2t}$$

  wählt, wobei t die Anzahl der zu korrigierenden Fehler darstellt. Diese Aussage werden wir im nächsten Unterkapitel noch etwas nutzen, um ein anderes und sogar schnelleres Decodierverfahren für BCH-Codes zu entwickeln.
- Goppa-Codes sind im allgemeinen nicht zyklisch aufgebaut.

3.8.3 Der BCH-Code als Sonderfall des Goppa-Codes

Wir haben gesehen, wie man mit dem Verfahren von Goppa einen Code zur Korrektur von t Fehlern aufbauen kann. Das Kernstück ist dabei das erzeugende Polynom c(z), das man innerhalb weiter Grenzen frei wählen darf, solange es einen Grad von mindestens t besitzt und außerdem keine Nullstellen im gerade benutzten Erweiterungskörper aufweist. Diese Freizügigkeit muss man für den Zweck der Fehlerkorrektur allerdings als Schwäche betrachten, da keine eindeutigen Vorschriften bekannt sind, um ein Polynom für einen möglichst leistungsfähigen, spar-samen Code zu finden, wie es etwa das Generatorpolynom des BCH-Codes bietet. Deshalb bleibt der praktische Wert des Goppa-Codes in diesem Zusammenhang z. Z. gering.

Auch der bereits gegebene Hinweis, dass sich der BCH-

Turbo-Lösung

Code als Sonderfall des Goppa-Codes ergibt, wenn man als erzeugendes Polynom $c(z) = z^{2t}$ wählt, hilft nicht weiter, da wir mit der Codewörter-Erzeugung über das Generatorpolynom bereits ein sehr vorteilhaftes Verfahren gefunden haben. Einzig die Möglichkeit, zum "schnellen" Decodieren den Euklidischen Algorithmus nutzen zu können, lohnt es, den BCH-Code nochmals unter diesem Blickwinkel zu betrachten. Um hierfür ein geschlossenes Verständnis zu erreichen, musste dazu allerdings ein ziemlicher Umweg eingeschlagen werden.

Der Erzeuger und sein Produkt

Überzeugen wir uns zunächst an einem unserer Beispiele, dass die Behauptung

> "das Polynom $c(z) = z^{2t}$ ist ein erzeugendes Polynom des Goppa-Codes, welches einen BCH-Code aufbaut"

tatsächlich stimmt. Wir bleiben dazu im $GF(2^4)$ und nehmen wieder einen 3-Fehler-korrigierbaren Code, wobei nun $c(z) = z^6$ sein muss. Da die allgemeine Berechnungsvorschrift aus Kapitel 3.8.1, Seite 193, für die modularen inversen Polynome $h_i(z)$

$$h_i(z) = \frac{c(z) - c(\beta_i)}{z - \beta_i} \cdot c(\beta_i)^{-1}, \quad i = 1, 2, \ldots\ldots, 2^m.$$

auch das Element $\beta_1 = 0$ enthält, $c(0)$ aber ebenfalls den Wert 0 annimmt, muss man auf das Polynom $h_1(z)$ verzichten, da es in diesem Fall nicht definiert ist.

$$\begin{aligned}
h_2 &= \beta^0 z^5 + \beta^0 z^4 + \beta^0 z^3 + \beta^0 z^2 + \beta^0 z + \beta^0 \\
h_3 &= \beta^9 z^5 + \beta^{10} z^4 + \beta^{11} z^3 + \beta^{12} z^2 + \beta^{13} z + \beta^{14} \\
h_4 &= \beta^3 z^5 + \beta^5 z^4 + \beta^7 z^3 + \beta^9 z^2 + \beta^{11} z + \beta^{13} \\
h_5 &= \beta^{12} z^5 + \beta^0 z^4 + \beta^3 z^3 + \beta^6 z^2 + \beta^9 z + \beta^{12} \\
h_6 &= \beta^6 z^5 + \beta^{10} z^4 + \beta^{14} z^3 + \beta^3 z^2 + \beta^7 z + \beta^{11} \\
h_7 &= \beta^0 z^5 + \beta^5 z^4 + \beta^{10} z^3 + \beta^0 z^2 + \beta^5 z + \beta^{10} \\
h_8 &= \beta^9 z^5 + \beta^0 z^4 + \beta^6 z^3 + \beta^{12} z^2 + \beta^3 z + \beta^9 \\
h_9 &= \beta^3 z^5 + \beta^{10} z^4 + \beta^2 z^3 + \beta^9 z^2 + \beta^1 z + \beta^8 \\
h_{10} &= \beta^{12} z^5 + \beta^5 z^4 + \beta^{13} z^3 + \beta^6 z^2 + \beta^{14} z + \beta^7 \\
h_{11} &= \beta^6 z^5 + \beta^0 z^4 + \beta^9 z^3 + \beta^3 z^2 + \beta^{12} z + \beta^6 \\
h_{12} &= \beta^0 z^5 + \beta^{10} z^4 + \beta^5 z^3 + \beta^0 z^2 + \beta^{10} z + \beta^5 \\
h_{13} &= \beta^9 z^5 + \beta^5 z^4 + \beta^1 z^3 + \beta^{12} z^2 + \beta^8 z + \beta^4
\end{aligned}$$

$$h_{14} = \beta^3 z^5 + \beta^0 z^4 + \beta^{12} z^3 + \beta^9 z^2 + \beta^6 z + \beta^3$$
$$h_{15} = \beta^{12} z^5 + \beta^{10} z^4 + \beta^8 z^3 + \beta^6 z^2 + \beta^4 z + \beta^2$$
$$h_{16} = \beta^6 z^5 + \beta^5 z^4 + \beta^4 z^3 + \beta^3 z^2 + \beta^2 z + \beta^1.$$

Tabelle 3-27: *Die 15 modularen inversen Polynome im GF(2^4) zu c(z) = z^6*

schöne Ausbeute

Es bleiben 15 modulare inverse Polynome übrig, deren mit den Codewortelementen v_i gewichtete Summe die Paritätsgleichung

$$\sum_{i=1}^{16} v_i \cdot h_i(z) = 0$$

erfüllen soll. Die 15 modularen inversen Polynome lässt man sich am einfachsten über ein kleines Programm berechnen, das Ergebnis ist in Tabelle 3-27 angegeben.

Verschwinden Sie!

Damit die Paritätsgleichung erfüllt werden kann, muss die mit den Elementen v_i gewichtete Polynomsumme für jeden ihrer 6 Polynomkoeffizienten verschwinden. Zur Verbesserung der Übersichtlichkeit "übersetzt" man die Elemente des GF(2^4) in die gleichwertigen Binärpolynome nach Bild 3.21, Seite 191, und erhält die 24 Paritätsgleichungen in Matrixform gemäß Bild 3.23 (für jeden der 6 Polynomkoeffizienten 4 binäre Gleichungen).

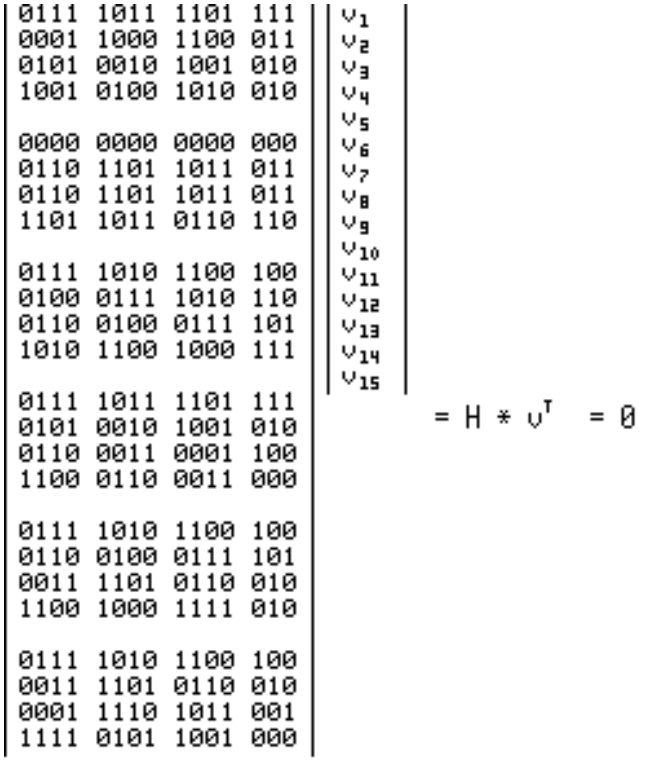

Bild 3.23: *Paritätsgleichungen des Goppa-Codes mit c(z) = z^6*

Lösungsmittel ....

Dies erscheint ziemlich aufwändig, wenn man es mit dem 3-Fehler-korrigierbaren Goppa-Code aus Kapitel 3.8.1 vergleicht. Hier hatten sich nur 16 Paritätsgleichungen ergeben. Wenn man allerdings die Matrix H durch elementare Zeilenoperationen auf eine rechte obere Dreiecksform bringt (Spaltenoperationen sind in diesem Beispiel nicht erforderlich), wie es in Bild 3.24 geschehen ist, dann sieht man, dass die meisten Gleichungen *linear abhängig* und damit überflüssig sind. Das Gleichungssystem lässt sich so auf genau 10 Gleichungen vermindern. In der Darstellung mit den Elementen des GF(2^4) erkennt man das nicht, da die Elemente jeweils zwar 4 Gleichungen in der Binärdarstellung entsprechen, aber durch ihre innere Verkopplung nicht so freizügig "auseinander gezogen" werden können.

Bitte endlich festlegen!

Es bleibt also die im Bild 3.25 dargestellte (10·15)-Paritätsmatrix H'', mit der man die Generatormatrix G'' des Codes aufbaut. Die Überlegungen dazu sind die glei-

chen wie schon in Kapitel 3.8.1: Bei 10 unabhängigen Gleichungen und 15 Codewortelementen können 5 Elemente frei vorgegeben werden, um die restlichen 10 festzulegen. Mit diesen 5 Elementen lassen sich 32 Codewörter berechnen.

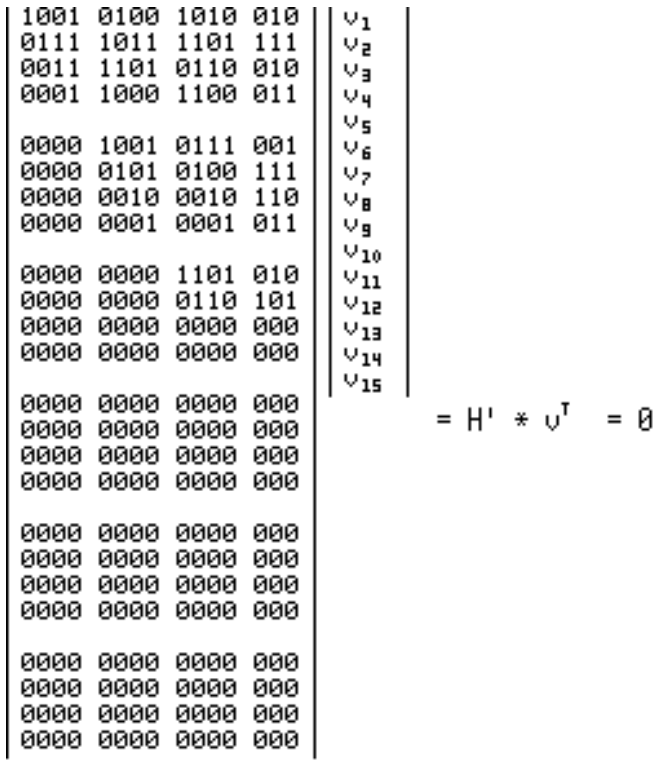

Bild 3.24: *Paritätsgleichungen aus Bild 3.23, auf rechte obere Dreiecksform reduziert*

Da die Matrix H'' bereits in Dreiecksform vorliegt, ist das Vorgehen ganz einfach. Die ersten 10 Elemente v_1, ..., v_{10} sind die festzulegenden, die letzten 5 v_{11}, ..., v_{15} werden vorgegeben. Man beginnt in der letzten Zeile, in der nur das festzulegende Element v_{10} steht. Daraus ergibt sich

$$v_{10} = (v_{11} + v_{13} + v_{15}) \text{ MOD } 2.$$

In Zeile 9 kommt v_9 hinzu,

$$v_9 = (v_{10} + v_{12} + v_{14}) \text{ MOD } 2$$
$$= (v_{11} + v_{12} + v_{13} + v_{14} + v_{15}) \text{ MOD } 2$$

usw.

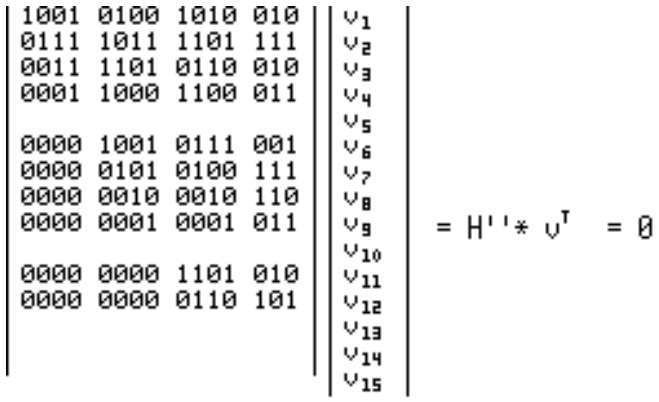

Bild 3.25: *Die auf 10 Gleichungen reduzierten Paritätsbeziehungen*

Stufenplan

Damit baut man schrittweise ein Gleichungssystem für die Codewortelemente v_1, ..., v_{10} auf, was sich am übersichtlichsten anhand der Generatormatrix G'' sehen lässt, siehe Bild 3.26. Da jede Zeile der Generatormatrix, bzw. jede Spalte der transponierten Generatormatrix G''T) ein linear unabhängiger Codewortvektor ist, sind damit nebenbei schon 5 Basis-Codewörter gefunden, z. B.:

$v_1 = u_1 \cdot G'' = [0\,0\,0\,0\,1] \cdot G'' = [0\,1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,0\,1]$

$v_2 = u_2 \cdot G'' = [0\,0\,0\,1\,0] \cdot G'' = [1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,1\,0]$

usw. Weitere Codewörter entstehen daraus als Linearkombinationen. Natürlich erfüllen alle die Paritätsbeziehung $H \cdot v^T = H' \cdot v^T = H'' \cdot v^T = 0$. Insbesondere gilt deshalb auch $H'' \cdot G''^T = 0$. Solche Codewörter sind etwa

$v_3 = u_3 \cdot G'' = [0\,0\,0\,1\,1] \cdot G'' = [1\,1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,0\,1\,1]$

$v_5 = u_5 \cdot G'' = [0\,0\,1\,0\,1] \cdot G'' = [0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,0\,1\,0\,1]$

$v_7 = u^7 \cdot G'' = [0\,0\,1\,0\,1] \cdot G'' = [1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,0\,1\,1\,1]$

Enge Verwandte

Wenn man diese Wörter mit denjenigen vergleicht, die über das BCH-Code-Verfahren in Unterkapitel 3.7.2 berechnet wurden, so stellt man fest, dass der Code zwar nicht dieselbe Zuordnung zu den Informationswörtern hat, die Menge aller Codewörter aber insgesamt gleich ist. Vor allem ist dieser Sonderfall des Goppa-Codes *zyklisch*.

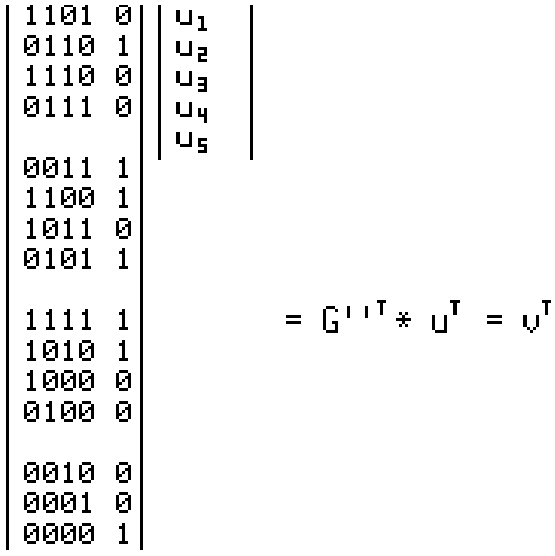

Bild 3.26: Generatormatrix für den (15,5)-Goppa-Code

Beide Aussagen prüfen wir stichprobenweise. So geht z. B. v_2 durch einen zyklischen Links-Schiebevorgang aus v_1 hervor, v_5 nach einem weiteren Schiebevorgang aus v_2. Das Codewort v_7 entsteht seinerseits durch zyklische Rechtsverschiebung aus v_3 usw.

Langer Weg, Ziel in Sichtweite

Nachdem jetzt an einem Beispiel nachvollzogen wurde, dass der Goppa-Code im GF(2^4) mit dem erzeugenden Polynom $c(z) = z^6$ tatsächlich den 3-Fehler-korrigierbaren BCH-Code im GF(2^4) ergibt, kann man versuchen, das angekündigte schnelle Decodierverfahren zu finden, dass auf der Lösung der schon verwendeten Beziehung

$$s(z)\cdot\sigma(z) \text{ MOD } c(z) = \sigma'(z)$$

aufbaut. Das Syndrompolynom s(z) hat hier den Grad 5 und damit 6 Koeffizienten s_0, ..., s_6, die sich durch Auswertung der Paritätsgleichung nach Bild 3.24 berechnen lassen. Wegen der anderen Darstellungsweise in Matrixform gilt hier zunächst

$$\begin{aligned} H\cdot w^T &= H\cdot (v+e)^T \\ &= H\cdot v^T + H\cdot e^T \\ &= 0 + H\cdot e^T = s^T \end{aligned}$$

mit dem Syndromvektor s. Dieser Vektor hat 2^4 Elemente, denen man von links nach rechts in 4-er Gruppen die 6 Polynomkoeffizienten von s(z) zuordnet. Ein Beispiel:

$$\begin{aligned} v &= v_7 \\ &= [1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,0\,1\,1\,1] \\ e &= [0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0] \\ w &= v_7 + e \\ &= [1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,0\,0\,1\,1\,1]. \end{aligned}$$

Das ergibt den Syndromvektor s und mit Hilfe der bewährten Tabelle 3-20 auf Seite 145 das Syndrompolynom s(z)

$$\begin{aligned} s &= [1101\quad 0000\quad 1101\quad 1001\quad 1001\quad 1011] \\ s(z) &= \beta^{13} z^5 + 0\, z^4 + \beta^{13} z^3 + \beta^{14} z^2 + \beta^{14} z^1 + \beta^7 z^0. \end{aligned}$$

In Unterkapitel 3.8.2 wurde festgestellt, dass der Grad des Fehlerpositions-Polynoms $\sigma(z)$ und der Grad seiner Ableitung $\sigma'(z)$ immer in bestimmten, von einander abhängigen Wertebereichen liegen müssen, damit eine Lösung für die genannte Gleichung überhaupt möglich ist. Hier liegen die Verhältnisse so:

Ein Weg für alle Fälle

- Sind 3 Fehler aufgetreten, dann hat das Fehlerpositions-Polynom wegen $\sigma(z) = z^3 + \sigma_2 z^2 + \sigma_1 z + \sigma_0$ auf jeden Fall den Grad 3 und die Ableitung $\sigma'(z)$ den Grad 2. Bei letzterer verschwindet außerdem immer der Koeffizient bei z^1, da die Ableitung einer geraden Potenz im GF(2^m) ein Vielfaches von 2 als Faktor enthält, was MOD 2 stets 0 ist.
- Sind 2 Fehler aufgetreten, dann hat $\sigma(z)$ den Grad 2 und $\sigma'(z)$ den Grad 0.
- Ist 1 Fehler aufgetreten, dann hat $\sigma(z)$ den Grad 1 und $\sigma'(z)$ den Grad 0.

Der EA: Nie war er so wertvoll wie heute

Mit dieser Kenntnis durchläuft man nun schrittweise den Euklidischen Algorithmus:

$$\begin{aligned} c(z) &= q_1(z)\cdot s(z) + r_2(z) \\ s(z) &= q_2(z)\cdot r_2(z) + r_3(z) \\ r_1(z) &= q_3(z)\cdot r_3(z) + r_4(z) \\ &\;\;\vdots \end{aligned}$$

usw., bis sich ein Restpolynom $r_i(z)$ ergibt, welches eine der drei genannten Bedingungen für den Grad der Ableitung $\sigma'(z)$ des Fehlerpositions-Polynoms aufweist (der Rechnungsgang selbst wurde in Kapitel 3.8.2 erläutert). Dann rechnet man mit den bis dahin ermittelten Multiplikatorpolynomen $q_j(z)$ (mit j = i-1, i-2, ...,2, 1) in der schon beschriebenen Weise zurück, bis man auf die Gleichung

$$\begin{aligned} r_i(z) &= \sigma'(z) \\ &= f\,[q_1(z), q_2(z), ..., q_{i-1}(z)]\cdot s(z) \\ &+ g\,[q_1(z), q_2(z), ..., q_{i-1}(z)]\cdot c(z) \end{aligned}$$

kommt. Die Modulbildung bezüglich c(z) führt auf

$$f\,[q_1(z), q_2(z), ..., q_{i-1}(z)]\cdot s(z) \text{ MOD } c(z) = r_i(z)$$

und deshalb ist der Polynomausdruck f(z) bei s(z)

$$f\,[q_1(z), q_2(z), ..., q_{i-1}(z)] = \sigma(z),$$

wenn eine der oben angegebenen Bedingungen bezüglich des Grades erfüllt wird.

Universell: Schieberegister

Der *Vorteil* dieses Lösungsweges liegt einmal darin, dass sich der Euklidische Algorithmus hier rein numerisch abarbeiten lässt, zum anderen die Auswertung besonders einfach mit Hilfe eines Schieberegisters erfolgen kann, welches nach den Rechenregeln des $GF(2^4)$ arbeitet. Ein solches Schieberegister ist zwar aufwändiger, jedoch nicht schwieriger als im Z_2.

Führen wir nun unser begonnenes Beispiel weiter. Der Euklidische Algorithmus startet bei der Gleichung

$$c(z) = z^6 = (\beta^2 z + 0)\cdot s(z) + (\beta^0 z^4 + \beta^1 z^3 + \beta^1 z^2 + \beta^9 z^1 + 0)$$

und wird mit den Schritten

$$\begin{aligned} s(z) &= (\beta^{13} z + \beta^{14})\cdot r_2(z) + (\beta^8 z^3 + \beta^4 z^2 + \beta^6 z^1 + \beta^7) \\ r_2(z) &= (\beta^7 z + \beta^{13})\cdot r_3(z) + (\beta^7 z^2 + 0z^1 + \beta^5) \end{aligned}$$

fortgeführt. Die Betrachtung des Restes $r_4(z) = \beta^7 z^2 + 0z^1 + \beta^5$ zeigt, dass die Bedingung für 3 Fehler aufgetreten sein könnte, da der Grad der Ableitung $\sigma'(z)$ dann 2 wäre und außerdem der Koeffizient bei z^1 gleich 0 ist. Die Rückrechnung ergibt

$$\begin{aligned} \sigma(z) &= (q_1(z) + q_1(z)\cdot q_2(z)\cdot q_3(z) + q_3(z)) \\ &= \beta^7 z^3 + \beta^3 z^2 + \beta^5 z^1 + \beta^{13} \end{aligned}$$

und $r_4(z)$ ist offensichtlich die Ableitung hierzu. Also liegt eine Lösung der Gleichung

$$s(z)\cdot\sigma(z) \text{ MOD } c(z) = \sigma'(z)$$

keine Fehlstellen mit Nullstellen

vor (diese Lösung ist sogar eindeutig), mit der die Nullstellen des Fehlerpositions-Polynoms ermittelt werden können. Diese sind

$$\beta_i = \beta^1$$
$$\beta_j = \beta^2$$
$$\beta_k = \beta^3.$$

Ein Weg für alle Fälle

Wie sieht es aus, wenn nur 2 Fehler aufgetreten sind? Nehmen wir als Beispiel

$$e = [0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0]$$

(wobei es unnötig ist, hierauf noch einen Codewortvektor v zu addieren, da der von diesem erzeugte Anteil im Syndromvektor ja ohnehin verschwindet). Das ergibt folgenden Syndromvektor s bzw. folgendes Syndrompolynom s(z):

$$s = [0000\ \ 0111\ \ 0101\ \ 0000\ \ 0011\ \ 0100]$$
$$s(z) = 0z^5 + \beta^{10}z^4 + \beta^8 z^3 + 0z^2 + \beta^4 z^1 + \beta^2 z^0.$$

Speist man damit den Euklidischen Algorithmus, so erhält man im ersten Schritt

$$c(z) = (\beta^5 z^2 + \beta^3 z + \beta^1)\cdot s(z) + \beta^3.$$

Beste Bedingungen

Eine Überprüfung ergibt, dass von den 3 oben genannten Fehlerbedingungen diejenige für 2 Fehler bereits in diesem ersten Schritt erfüllt ist. Das Fehlerpositions-Polynom lautet demnach

$$\sigma(z) = \beta^5 z^2 + \beta^3 z + \beta^1$$

und die Nullstellen sind

$$\beta_i = \beta^3$$
$$\beta_j = \beta^8.$$

Nun noch ein Beispiel, wo der 1-Bit-Fehler

$$e = [0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0]$$

auftrat. Er führt auf folgenden Syndromvektor s bzw. auf folgendes Syndrompolynom s(z):

$$s = [0001\ \ 0110\ \ 0111\ \ 0000\ \ 0110\ \ 0111]$$
$$s(z) = \beta^0 z^5 + \beta^5 z^4 + \beta^{10} z^3 + \beta^0 z^2 + \beta^5 z^1 + \beta^{10} z^0.$$

... und immer wieder: EA

Auch hier liefert bereits der erste Schritt des Euklidischen Algorithmus die Lösung:

$$c(z) = (\beta^0 z + \beta^5)\cdot s(z) + \beta^0.$$

Von den 3 Fehlerbedingungen trifft offenbar diejenige für einen 1-Bit-Fehler zu. Das Fehlerpositions-Polynom lautet demnach

$$\sigma(z) = z + \beta^5$$

mit der Nullstelle

$$\beta_i = \beta^5$$

Zum Abschluss eine letzte Frage: Was passiert, wenn mehr als 3 Fehler aufgetreten sind. Hier lassen sich folgende Fälle unterscheiden:

Zu viele Fehler: Ende der Fahnenstange

- Einige Fehleranordnungen können das gesendete Codewort v so verändern, dass ein anderes gültiges Codewort v' entsteht, nämlich dann, wenn der Fehler e zufällig selbst ein Codewort ist. Die Linearkombination zweier Codewörter ergibt wegen deren Zugehörigkeit zu einer mathematischen Gruppe immer ein weiteres Codewort (siehe Unterkapitel 3.4.1). Anhand unserer Beispiele ist das gut zu erkennen. So ergibt die Summe von v_1 und v_2 das Codewort v_3:

  $$\begin{aligned} v_1 + v_2 &= [0\,1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,0\,1] \\ &+ [1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,0\,1\,0] \\ &= [1\,1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,0\,1\,1] = v_3. \end{aligned}$$

  Kein Decodierverfahren der Welt ist dann in der Lage, dies zu erkennen. Wie oft kann so etwas eigentlich eintreten? Bei unserem (15,5)-Code sind 2^{15} Empfangswörter möglich, aber nur 2^5 als Codewörter zulässig. Nimmt man das Nullwort aus, das natürlich keinen wirklichen Fehler darstellt, so bleiben für e genau $2^5 - 1$ Möglichkeiten. Da alle denkbaren Fehler ihrerseits in $2^{15} - 1$ Varianten auftreten können, beträgt die theoretische Wahrscheinlichkeit $(2^5-1)/(2^{15}-1) \approx 2^{-10}$, ist also kleiner als 1/1000. Die praktische Wahrscheinlichkeit sollte allerdings bedeutend geringer sein, was durch die Technologie sicherzustellen wäre. Anders ausgedrückt wird man den Code so wählen, dass das Auftreten von mehr Fehlern, als mit dem Verfahren korrigierbar, so gut wie ausgeschlossen bleibt.
- Weiterhin kann es sein, dass dem Decodierverfahren

Täuschungsversuch

"vorgetäuscht" wird, es wären Fehler in einer korrigierbaren Anzahl aufgetreten. Dieser Fall tritt ein, wenn sich der Fehler so interpretieren lässt, als wäre er aus einem zulässigen Codewort v' und einer korrigierbaren Anzahl von Einzelfehlern entstanden. Dann baut sich das Empfangswort w aus

$$w = v + e = v + (v' + e') = v'' + e'$$

auf und wird fälschlicherweise als Codewort v'' decodiert. (Übrigens hat der Fehler e dann ein Gewicht von wenigstens 4 (also 4 Einsen), denn jedes Codewort (außer dem Nullwort) enthält mindestens 7 Einsen (Mindest-Hammingabstand für 3-Fehler-Korrigierbarkeit $d = 2 \cdot 3 + 1 = 7$), was bei Addition eines 3-Bit-Fehlers im ungünstigsten Fall 4 Einsen ergibt, meistens natürlich mehr).

- Alle anderen Fehler führen bei der Verwendung des daraus entstehenden Syndrompolynoms s(z) im Euklidischen Algorithmus zwar auf eine der oben genannten verträglichen Fehlerbedingungen, das Fehlerpositions-Polynom σ(z) hat im GF(2^4) dann aber keine Nullstellen, so dass sich Fehler nicht korrigieren lassen. Ein Beispiel mag diesen Sachverhalt ein wenig erläutern. Wir gehen von einem 4-Bit-Fehler

  $e = [0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0]$

  aus, der den Syndromvektor s und das Syndrompolynom s(z)

  $s = [0001 \quad 0111 \quad 0100 \quad 0001 \quad 0010 \quad 0101]$

  $s(z) = \beta^0 z^5 + \beta^{10} z^4 + \beta^2 z^3 + \beta^0 z^2 + \beta^1 z^1 + \beta^8 z^0.$

  erzeugt. Der Euklidische Algorithmus wird also mit

  $c(z) = z^6 = (\beta^0 z + \beta^{10}) \cdot s(z) + (\beta^1 z^4 + \beta^{11} z^3 + \beta^8 z^2 + \beta^7 z^1 + \beta^3)$

  gestartet und setzt sich mit

  $s(z) = (\beta^{14} z + 0) \cdot r_2(z) + (\beta^{12} z^3 + \beta^{13} z^2 + \beta^5 z^1 + \beta^8)$

  $r_2(z) = (\beta^4 z + \beta^{12}) \cdot r_3(z) + (\beta^3 z^2 + 0\, z^1 + \beta^{11})$

  fort. Im dritten Schritt ergeben sich für

  $\sigma(z) = (q_1(z) + q_1(z) \cdot q_2(z) \cdot q_3(z) + q_3(z)$

  $= \beta^3 z^3 + \beta^4 z^2 + \beta^{11} z + \beta^3$

  $\sigma'(z) = r_4(z) = \beta^{14}$

Mehr Fehler: manchmal erkennbar

verträgliche Bedingungen, so dass man annehmen könnte, es seien 3 Fehler aufgetreten. Keines der 15 nutzbaren Elemente (0 ausgeschlossen) ist aber eine Nullstelle des Fehlerpositions-Polynoms σ(z). Daraus schließt man das Auftreten von mehr als 3 Fehlern.

Insgesamt steht jetzt ein Verfahren zur Decodierung von BCH-Codes zur Verfügung, welches tatsächlich besonders einfach arbeitet und dem im Unterkapitel 3.7.7 beschriebenen überlegen ist. Einen kleinen Wermutstropfen muss man dennoch schlucken: Bei einem BCH-Code zur Korrektur von t Fehlern sind hier doppelt so viele Syndromwerte zu berechnen, ob man das nun über die Paritätsmatrix H oder direkt aus den Paritätsgleichungen im GF(2^m) heraus macht. Trotzdem werden die Vorteile überwiegen. *Adamek* erläutert übrigens im Kapitel 13 seines Buches das Verfahren in allgemeiner Form. Wir begnügen uns mit den geschilderten Grundgedanken, die bereits wesentliche Gesichtspunkte enthalten.

Reed-Muller-Code

Nach all den doch sehr mathematisch geprägten Gedanken zur Erzeugung und Decodierung von zyklischen Codes und Goppa-Codes soll nun noch – gewissermaßen zur Entspannung – ein Blick auf einen vollkommen anders arbeitenden Code geworfen werden, der mit einfachster Mathematik auskommt und dennoch recht wirkungsvoll ist, den Reed-Muller-Code. Er geht von einer direkt aufstellbaren Generatormatrix G aus. Die Decodierung der Empfangswörter erfolgt nicht etwa über das vertraute Syndrom, sondern über eine andere pfiffige Technik. Daher gibt es beim Reed-Muller-Code auch keine Paritätsbeziehungen im bisher gebrauchten Sinn.

Schauen wir uns zunächst die Erzeugung der Codewörter an und wählen dazu einen Code zur Korrektur von maximal 3 Fehlern. Hierfür eignet sich ein solcher der Länge $n = 2^4 = 16$. Die Generatormatrix besteht aus 5 Zeilen, die einen ganz charakteristischen Aufbau haben. Man betrachtet jede Zeile als Zeilenvektor, der mit r bezeichnet und mit dem Index 0 beginnend durchnummeriert wird. Der Erste, also r_0, soll lauter "1"-Elemente haben, der