Inhaltsverzeichnis

Kapitel 1

Reed-Solomon-Code

Joshua Bär und Michael Steiner

1.1 Einleitung

Der Reed-Solomon-Code ist entstanden um, das Problem der Fehler bei der Datenübertragung, zu lösen. In diesem Abschnitt wird möglichst verständlich die mathematische Abfolge, Funktion oder Algorithmus des Reed-Solomon-Code erklärt. Es wird jedoch nicht auf die technische Umsetzung oder Implementierung eingegangen.

1.2 Idee

Um beim Datenübertragen Fehler zu erkennen, könnte man die Daten jeweils doppelt senden, und so jeweilige Fehler zu erkennen. Doch nur schon um Fehler zu erkennen werden überproportional viele Daten doppelt und dreifach gesendet. Der Reed-Solomon-Code macht dies auf eine andere, clevere Weise. Das Problem liegt darin Informationen, Zahlen, zu Übertragen und Fehler zu erkennen. Speziell beim Reed-Solomon-Code kann man nicht nur Fehler erkennen, man kann sogar einige Fehler korrigieren. Der Unterschied des Fehler erkennen und korrigiren, ist das beim Erkennen nur die Frage beantwortet wird: Ist die Übertragung fehlerhaft oder nicht? Beim Korrigieren werden Fehler erkannt und dann zusätzlich noch den original Wert rekonstruieren. Auch eine Variante wäre die Daten nach einer Fehlerhaften sendung, nochmals zum senden auffordern(auch hier wird doppelt und dreifach gesendung), was bei Reed-Solomon-Code-Anwendungen nicht immer sinnvoll ist. 1.11

1.2.1 Polynom-Ansatz

Eine Idee ist, aus den Daten ein Polynom zu bilden. Diese Polynomfunktion bei bestimmten Werten errechnet und diese Punkte dann überträgt.

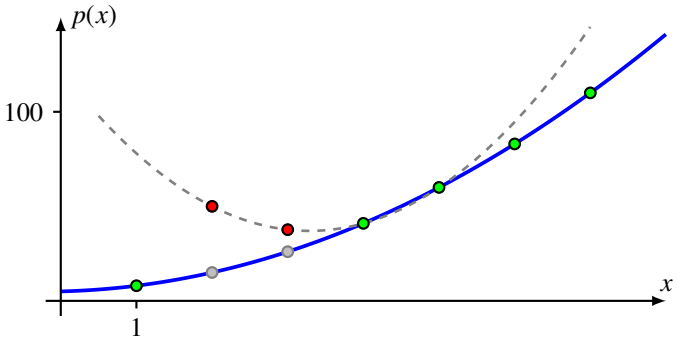


Abbildung 1.1: Polynom $p(x)$ von der Gleichung(1.1)

Beispiel. Nehmen wir die Zahlen 2, 1, 5, welche uns dann das Polynom

$$p(x) = 2x^2 + 1x + 5 \tag{1.1}$$

ergeben. Übertragen werden nun die grünen Werte dieses blauen Polynomes an den Stellen 1, 2, 3...7 dieses Polynomes. Grafisch sieht man dies dann in Abbildung 1.1, mit den Punkten, $p(1)$, $p(2)$, ..., $p(7) =$ (8, 15, 26, 41, 60, 83, 110) Wenn ein Fehler sich in die Übertragung eingeschlichen hat, muss der Leser/Empfänger diesen erkennen und das Polynom rekonstruieren. Der Leser/Empfänger weiss, den Grad des Polynoms und dessen Werte übermittelt wurden. Die Farbe blau brauchen wir für die Daten welche wir mit der Farbe grün Übermitteln. ○

Beispiel. Ein Polynome zweiten Grades ist durch drei Punkte eindeutig bestimmbar. Hat es Fehler in der Übertragunge gegeben,in der Abbilbung 1.1 die roten Punkte). Erkennt man diese Fehler, da alle korrekten Punkte auf der Parabel liegen müssen. Die grünen Punkte bestimmen die Parabel, und die Fehler können zu den Orginalpunkte rekonstruiert werden. Ab wie vielen Fehler ist das Polynom nicht mehr erkennbar beim Übertragen von 7 Punkten? Bei 2 Fehlern kann man noch eindeutig bestimmen, dass das Polynom mit 4 Punkten, gegenüber dem mit 5 Punkten falsch liegt. 1.1 Werden es mehr Fehler kann nur erkannt werden, dass das Polynom nicht stimmt. Das orginale Polynom kann aber nicht mehr gefunden werden. Da andere Polynome oder das Konkurrenzpolynom, grau gestrichelt in Abbildung 1.1, das orginal fehlleitet. Um das Konkurrenzpolynom auszuschliessen, währen mehr Übertragungspunkte nötig. ○

1.3 Fehlerkorekturstellen bestimmen

Um zu bestimmen wieviel zusätzliche Übertragungspunkte notwendig sind, um die Fehler zu korrigieren, muss man zuerst wissen, wieviel Daten gesendet und wieviel Fehler erkennt werden sollen. Die Anzahl Daten (ab hier verwenden wir das Wort Nutzlast), die als Polynomkoeffizente k übergeben werden, brauchen die gleiche Anzahl an Polynomkoeffizententräger, beginnend bei Grad 0 somit ergibt sich der Polynomgrad mit $k - 1$. Für die Anzahl der Fehler t, welche korrigiert werden können, gehen wir zum Beispiel.

Beispiel. von den Polynom 1.1 in, welchem wir 7 Übertragungspunkte senden. Durch 3 Punkte wird das Polyom eindeutig bestimmt, nun haben wir mehrere Konkurrenzpolynome, doch mit maximal 2

Nutzlas	Fehler	Übertragen
3	2	7 Werte eines Polynoms vom Grad 2
4	2	8 Werte eines Polynoms vom Grad 3
3	3	9 Werte eines Polynoms vom Grad 2
k	t	$k + 2t$ Werte eines Polynoms vom Grad $k - 1$

Tabelle 1.1: Fehlerkorrekturstellen Bestimmung.

Fehler liegen auf einem Konkurrenzpolynom, maximal 4 Punkte und auf unserem orginal 5 Punkte. Ansonsten hatt es mehr Fehler oder unser Konkurrenzpolynom ist das gleiche wie das Original. Somit können wir nun bestimmen, dass von den 7 Übertragungspunkten$u$ bis zu 2 Fehler korrigiert werden können und 4 Übertragungspunkte zusätzlich gesendet werden müssen. ○

Man könnte auch dies in der Tabelle 1.1 erkennen, doch mit dieser Gleichung

$$\frac{u - k}{t} = 2 \tag{1.2}$$

zeigt sich, dass es $k + 2t$ Übertragungspunkte braucht.

Ein Nebeneffekt ist, dass dadurch auch $2t$ Fehler erkannt werden können, nicht aber korrigiert. Um aus den übertragenen Zahlen wieder die Nutzlastzahlen zu bekommen könnte man eine Polynominterpolation anwenden, doch die Punkte mit Polynominterpolation zu einem Polynom zu rekonstruieren ist schwierig und fehleranfällig.

1.4 Übertragung mit Hilfe der Diskrten Fourientransformation

Um die Polynominterpolation zu umgehen, gehen wir nun über in die Fourietransformation. Dies wird weder eine Erklärung der Forientransorfmation, noch ein genauer gebrauch für den Reed-Solomon-Code. Dieser Abschnitt zeigt nur wie die Fourietransformation auf Fehler reagiert. Das ganze zeigen wir mit einem Beispiel einer Übertragung von Zahlen mit Hilfe der Fourietransformation.

1.4.1 Diskrete Fourietransformation Zusamenhang

Mit hilfe der Fourietransformation werden die blauen Datenpunkte transformiert, zu den grünen Übertragungspunkten. Durch eine Rücktransformation könnnen die blauen Datenpunkte wieder rekonstruiert werden.

Beispiel einer Übertragung

Der Auftrag ist nun 64 Daten zu übertragen und nach 32 Fehler abzusicheren, 16 Fehler erkennen und rekonstruieren.

Dieser Auftrag soll mittels Fouriertransformation bewerkstelligt werden. In der Abbildung 1.4.1 sieht man dies Schritt für Schritt, und hier werden die einzelne Schritte erklärt:

(1) Das Signal hat 64 die Daten k, hier zufällige Zahlen, welche übertragen werden sollen. Zusätzlich soll nach 16 Fehler t, die rekonstruierbar sind abgesichert werden. Das macht dann insgesamt $k + 2t = 64 + 2 \cdot 16 = 96$ Übertragungszahlen. (siehe Abschnitt 1.3) Die 32 Fehlerkorrekturstellen werden als Nullzahlen Übertragen.

(2) Nun werden mittels der diskreten Fourietransformation diese 96 codiert, transformiert. Das heisst alle Informationen ist in alle Zahlenvorhanden, auch die Fehlerkorrekturstellen Nullzahlen.

(3) Nun kommen drei Fehler dazu an den Übertragungsstellen 7, 21 und 75. Die Fehler können auf den ganzen 96 Übertragungswerten liegen, wie die 75 zeigt. Zu Beachten ist auch noch, dass der Fehler um das 20- bis 150-Fache kleiner ist.Die Fehlerskala ist rechts.

(4) Dieses wird nun Empfangen, man kann keine Fehler erkennen, da diese soviel kleiner sind. Für das Decodieren wird die Inverse Fourietransformation angewendet, und alle Fehler werden mittransformiert.

(5) Nun sieht man die Fehler im decodierten Signal in den Übertragungszahlen. Von den Übertragungsstellen 64 bis 96 erkennt man, das diese nicht mehr Null sind.

(6) Diese Fehlerkorrekturstellen 64 bis 96, dies definieren wir als Syndrom. In diesem Syndrom ist die Fehlerinformation gespeichert und muss nur noch transformiert werden.

(7) Hier sieht man genau wo die Fehler stattgefunden haben. Leider nicht mehr mit der Qualtiätt der Ursprünglichen Fehler, sie sind nur noch 0.6 oder 0.4 gross. Obwohl der Fehler um das 20Fache kleiner ist erkennt man im Locator die Fehlerstellen wieder.

Nun haben wir mit Hilfe der Fourietransformation die 3 Fehlerstellen durch das Syndrom lokalisiert, jetzt gilt es nur noch diese zu korrigieren und wir haben unser originales Signal wieder.

Nun zur definition der Diskrete Fourietransformation, diese ist definiert als

$$\hat{c}_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n \cdot e^{-\frac{2\pi j}{N} \cdot kn}., \tag{1.3}$$

Wenn man nun

$$w = e^{-\frac{2\pi j}{N} k} \tag{1.4}$$

ersetzte, und N konstantbleibt, erhält man

$$\hat{c}_k = \frac{1}{N}(f_0 w^0 + f_1 w^1 + f_2 w^2 + \cdots + f_{N-1} w^N) \tag{1.5}$$

was überaust ähnlich zu unserem Polynomidee ist. Die Polynominterpolation und die Fourietransformation rechnen beide mit reelen Zahlen. Wenn die Fehlerabweichung sehr sehr klein ist, erkennt man diese irgendwann nicht mehr. Zusätzlich muss mann immer Grenzen bestimmen auf wieviel Stellen gerechnet wird und wie die Fehler erkannt werden im Locator. Deshalb haben Mathematiker einen neuen Körper gesucht und ihn in der Endlichkeit gefunden, dies wird nun im nächsten Abschnitt genauer erklärt.

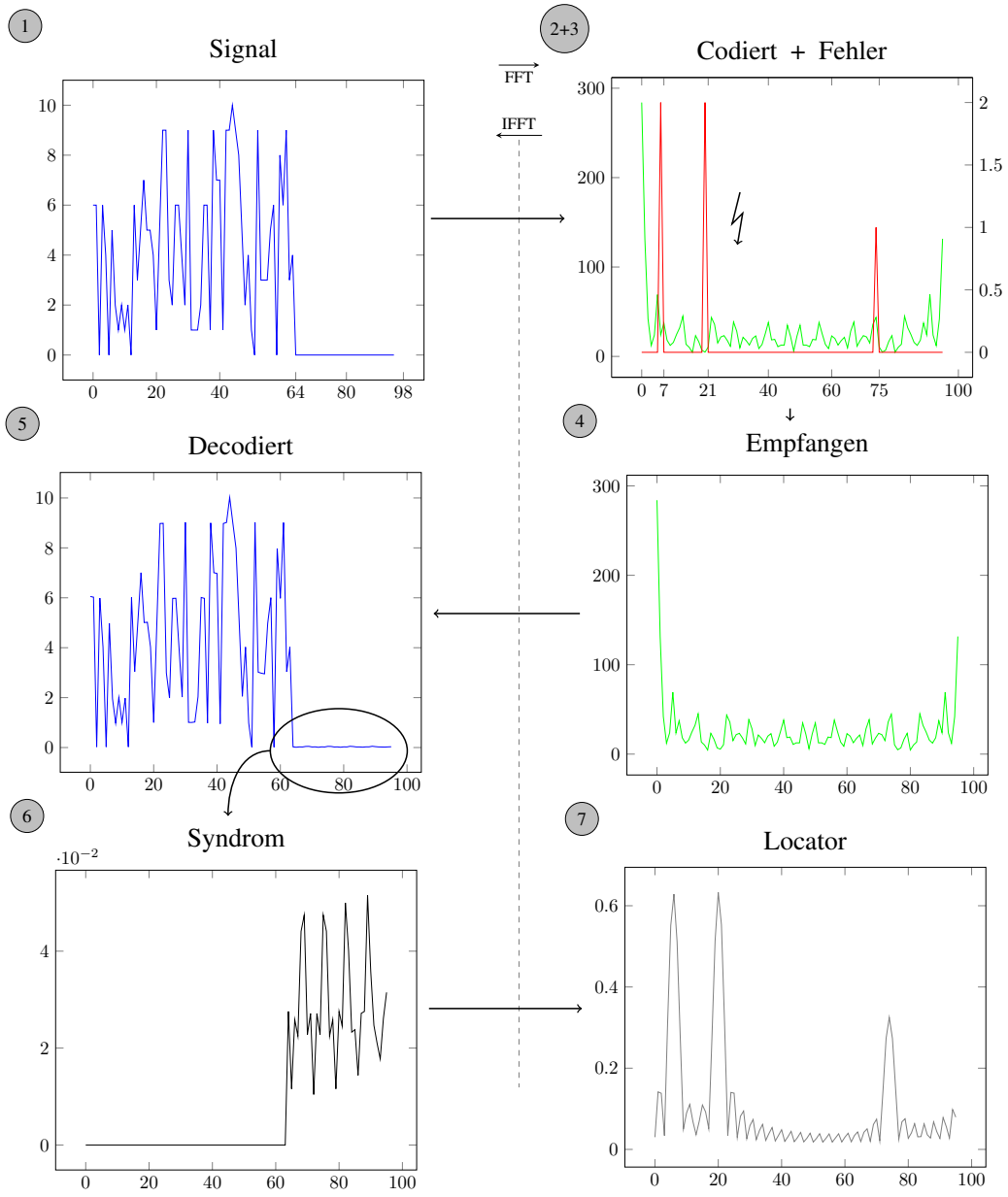


Abbildung 1.2: Übertragungsabfolge 1.4.1

1.5 Reed-Solomon in Endlichen Körpern

TODO: (warten auf den 1. Teil)

Das Rechnen in endlichen Körpern bietet einige Vorteile:

- Konkrete Zahlen: In endlichen Körpern gibt es weder rationale noch komplexe Zahlen. Zudem beschränken sich die möglichen Rechenoperationen auf das Addieren und Multiplizieren. Somit können wir nur ganze Zahlen als Resultat erhalten.
- Digitale Fehlerkorrektur: lässt sich nur in endlichen Körpern umsetzen.

Um jetzt eine Nachricht in den endlichen Körpern zu konstruieren legen wir fest, dass diese Nachricht aus einem Nutzdatenteil und einem Fehlerkorrekturteil bestehen muss. Somit ist die zu übertragende Nachricht immer grösser als die Daten, die wir übertragen wollen. Zudem müssen wir einen Weg finden, den Fehlerkorrekturteil so aus den Nutzdaten zu berechnen, dass wir die Nutzdaten auf der Empfängerseite wieder rekonstruieren können, sollte es zu einer fehlerhaften Übertragung kommen.

Nun stellt sich die Frage, wie wir eine fehlerhafte Nachricht korrigieren können, ohne ihren ursprünglichen Inhalt zu kennen. Der Reed-Solomon-Code erzielt dies, indem aus dem Fehlerkorrekturteil ein sogenanntes “Lokatorpolynom” generiert werden kann. Dieses Polynom gibt dem Empfänger an, welche Stellen in der Nachricht feherhaft sind.

1.6 Codierung eines Beispiels

Um die Funktionsweise eines Reed-Solomon-Codes besser zu verstehen, werden wir die einzelnen Probleme und ihre Lösungen anhand eines Beispiels betrachten. Da wir in endlichen Körpern rechnen, werden wir zuerst solch einen Körper festlegen. Dabei müssen wir die Definition **??** berücksichtigen, die besagt, dass nur Primzahlen für endliche Körper in Frage kommen. Wir legen für unser Beispiel den endlichen Körper $\mathbb{F}_q$ mit $q = 11$ fest. Zur Hilfestellung zum Rechnen in $\mathbb{F}_{11}$ können die beiden Tabellen 1.12.1 und 1.12.2 hinzugezogen werden. Diese Tabellen enthalten die Resultate der arithmetischen Operationen im Körper $\mathbb{F}_{11}$, die durchgeführt werden können. Aus der Definition der endlichen Körper (ersichtlich auch in den Tabellen) folgt, dass uns nur die Zahlen

$$\mathbb{F}_{11} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

zur Verfügung stehen und somit $11 = 0$ gelten muss.

Die Menge uns zur Verfügung stehender Zahlen legt auch fest, wie viele Zahlen ein Nachrichtenblock n, bestehend aus Nutzdatenteil und Fehlerkorrekturteil, umfassen kann. Der Nachrichtenblock im Beispiel besteht aus

$$n = q - 1 = 10 \text{ Zahlen},$$

wobei die null weggelassen wird. Wenn wir versuchen würden, mit der null zu codieren, so stellen wir fest, dass wir wieder null an der gleichen Stelle erhalten und somit wäre die Codierung nicht eindeutig.

Im nächsten Schritt bestimmen wir, wie viele Fehler t maximal während der Übertragung auftreten dürfen, damit wir sie noch korrigieren können. Unser Beispielcode sollte in der Lage sein

$$t = 2$$

Fehlerstellen korrigieren zu können.

Die Grösse des Nutzdatenteils hängt von der Grösse des Nachrichtenblocks sowie der Anzahl der Fehlerkorrekturstellen ab. Je robuster der Code sein muss, desto weniger Platz für Nutzdaten k bleibt in der Nachricht übrig. Bei maximal 2 Fehler können wir noch

$$k = n - 2t = 6 \text{ Zahlen}$$

übertragen.

Zusammenfassend haben wir einen Nachrichtenblock mit der Länge von 10 Zahlen definiert, der 6 Zahlen als Nutzlast beinhaltet und in der Lage ist, aus 2 fehlerhafte Stellen im Block die ursprünglichen Nutzdaten zu rekonstruieren. Zudem werden wir im weiteren feststellen, dass dieser Code maximal vier Fehlerstellen erkennen, diese aber nicht rekonstruieren kann.

Wir legen nun für das Beispiel die Nachricht

$$m = [0, 0, 0, 0, 4, 7, 2, 5, 8, 1]$$

fest, die wir gerne an einen Empfänger übertragen möchten, wobei die vorderen vier Stellen für die Fehlerkorrektur zuständig sind. Solange diese Stellen vor dem Codieren und nach dem Decodieren den Wert null haben, so ist die Nachricht fehlerfrei übertragen worden.

Da wir in den folgenden Abschnitten mit Polynomen arbeiten, stellen wir die Nachricht auch noch als Polynom

$$m(X) = 4X^5 + 7X^4 + 2X^3 + 5X^2 + 8X + 1$$

dar.

1.6.1 Der Ansatz der diskreten Fouriertransformation

In einem vorherigen Abschnitt (???) haben wir schon einmal die diskrete Fouriertransformation zum Codieren einer Nachricht verwendet. In den endlichen Körpern wird dies jedoch nicht gelingen, da die Eulerische Zahl e in endlichen Körpern nicht existiert. Wir wählen deshalb eine Zahl a, die die gleichen Aufgaben haben soll wie $e^{\frac{j}{2\pi}}$ in der diskreten Fouriertransformation, nur mit dem Unterschied, dass a in $\mathbb{F}_{11}$ ist. Dazu soll die Potenz von a den gesamten Zahlenbereich von $\mathbb{F}_{11}$ abdecken. Dazu ändern wir die Darstellung von

$$\mathbb{F}_{11} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

in die von a abhängige Schreibweise

$$\mathbb{Z}_{11} \setminus \{0\} = \{a^0, a^1, a^2, a^3, a^4, a^5, a^6, a^7, a^8, a^9\}.$$

Die primitiven Einheitswurzeln

Wenn wir jetzt Zahlen von $\mathbb{F}_{11}$ an Stelle von a einsetzen, erhalten wir

$$
\begin{array}{lcllcl}
a = 1 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,1,1,1,1,1,1,1,1,1\} & \neq & \mathbb{F}_{11} \setminus \{0\} \\
a = 2 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,2,4,8,5,10,9,7,3,6\} & = & \mathbb{F}_{11} \setminus \{0\} \\
a = 3 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,3,9,5,4,1,3,9,5,4\} & \neq & \mathbb{F}_{11} \setminus \{0\} \\
a = 4 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,4,5,9,3,1,4,5,9,3\} & \neq & \mathbb{F}_{11} \setminus \{0\} \\
a = 5 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,5,3,4,9,1,5,3,4,9\} & \neq & \mathbb{F}_{11} \setminus \{0\} \\
a = 6 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,6,3,7,9,10,5,8,4,2\} & = & \mathbb{F}_{11} \setminus \{0\} \\
a = 7 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,7,5,2,3,10,4,6,9,8\} & = & \mathbb{F}_{11} \setminus \{0\} \\
a = 8 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,8,9,6,4,10,3,2,5,7\} & = & \mathbb{F}_{11} \setminus \{0\} \\
a = 9 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,9,4,3,5,1,9,4,3,5\} & \neq & \mathbb{F}_{11} \setminus \{0\} \\
a = 10 & \Rightarrow & \{a^i | 0 \leq i \leq 10\} & = & \{1,10,1,10,1,10,1,10,1,10\} & \neq & \mathbb{F}_{11} \setminus \{0\}.
\end{array}
$$

Es fällt auf, dass wir für a die Zahlen $2, 6, 7, 8$ Mengen erhalten, die tatsächlich den gesamten Zahlenraum von $\mathbb{F}_{11}$ abbilden. Solche Zahlen werden *primitive Einheitswurzel* genannt. Wenden wir diese Vorgehensweise auch für andere endliche Körper an, so werden wir sehen, dass wir immer mindestens zwei solcher Einheitswurzel finden werden. Somit ist es uns überlassen, eine dieser Einheitswurzel auszuwählen, mit der wir weiter rechnen wollen. Für das Beispiel wählen wir die Zahl $a = 8$.

Bildung einer Transformationsmatrix

Mit der Wahl einer Einheitswurzel ist es uns jetzt möglich, unsere Nachricht zu Codieren. Daraus sollen wir dann einen Übertragungsvektor v erhalten, den wir an den Empfänger schicken können. Für die Codierung setzen wir alle Zahlen in $\mathbb{F}_{11} \setminus \{0\}$ nacheinander in $m(X)$ ein. Da wir zuvor eine von a abhängige Schreibweise gewählt haben setzen wir stattdessen a^i ein mit $a = 8$ als die von uns gewählten primitiven Einheitswurzel. Daraus ergibt sich

$$
\begin{aligned}
m(8^0) &= 4 \cdot 1^5 + 7 \cdot 1^4 + 2 \cdot 1^3 + 5 \cdot 1^2 + 8 \cdot 1^1 + 1 = 5 \\
m(8^1) &= 4 \cdot 8^5 + 7 \cdot 8^4 + 2 \cdot 8^3 + 5 \cdot 8^2 + 8 \cdot 8^1 + 1 = 3 \\
&\vdots \\
m(8^9) &= 4 \cdot 7^5 + 7 \cdot 7^4 + 2 \cdot 7^3 + 5 \cdot 7^2 + 8 \cdot 7^1 + 1 = 4
\end{aligned}
$$

als unser Übertragungsvektor.

1.6.2 Allgemeine Codierung

Um das Ganze noch ein wenig übersichtlicher zu gestalten, können wir die Polynome zu einer Matrix zusammenfassen, die unsere Transformationsmatrix A bildet.

Für die allgemeine Codierung benötigen wir die Nachricht m, die codiert werden soll, sowie die Transformationsmatrix A. Daraus erhalten wir den Übertragungsvektor v. Setzen wir die Zahlen aus

dem Beispiel ein erhalten wir folgende Darstellung:

$$v = A \cdot m \qquad \Rightarrow \qquad v = \begin{pmatrix} 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 \\ 8^0 & 8^1 & 8^2 & 8^3 & 8^4 & 8^5 & 8^6 & 8^7 & 8^8 & 8^9 \\ 8^0 & 8^2 & 8^4 & 8^6 & 8^8 & 8^{10} & 8^{12} & 8^{14} & 8^{16} & 8^{18} \\ 8^0 & 8^3 & 8^6 & 8^9 & 8^{12} & 8^{15} & 8^{18} & 8^{21} & 8^{24} & 8^{27} \\ 8^0 & 8^4 & 8^8 & 8^{12} & 8^{16} & 8^{20} & 8^{24} & 8^{28} & 8^{32} & 8^{36} \\ 8^0 & 8^5 & 8^{10} & 8^{15} & 8^{20} & 8^{25} & 8^{30} & 8^{35} & 8^{40} & 8^{45} \\ 8^0 & 8^6 & 8^{12} & 8^{18} & 8^{24} & 8^{30} & 8^{36} & 8^{42} & 8^{48} & 8^{54} \\ 8^0 & 8^7 & 8^{14} & 8^{21} & 8^{28} & 8^{35} & 8^{42} & 8^{49} & 8^{56} & 8^{63} \\ 8^0 & 8^8 & 8^{16} & 8^{24} & 8^{32} & 8^{40} & 8^{48} & 8^{56} & 8^{64} & 8^{72} \\ 8^0 & 8^9 & 8^{18} & 8^{27} & 8^{36} & 8^{45} & 8^{54} & 8^{63} & 8^{72} & 8^{81} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 8 \\ 5 \\ 2 \\ 7 \\ 4 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Für unseren Übertragungsvektor resultiert

$$v = [5, 3, 6, 5, 2, 10, 2, 7, 10, 4],$$

den wir jetzt über einen beliebigen Nachrichtenkanal versenden können.

1.7 Decodierung: Ansatz ohne Fehler

In diesem Abschnitt betrachten wie die Überlegung, wie wir auf der Empfängerseite die Nachricht aus dem empfangenen Übertragungsvektor erhalten. Nach einer einfachen Überlegung müssen wir den Übertragungsvektor decodieren, was auf den ersten Blick nicht allzu kompliziert sein sollte, solange wir davon ausgehen können, dass es während der Übertragung keine Fehler gegeben hat. Wir betrachten deshalb den Übertragungskanal als fehlerfrei.

Der Übertragungsvektor empfangen wir also als

$$v = [5, 3, 6, 5, 2, 10, 2, 7, 10, 4].$$

Nach einem banalen Ansatz ist die Decodierung die Inverse der Codierung. Dank der Matrixschreibweise lässt sich dies relativ einfach umsetzen.

$$v = A \cdot m \qquad \Rightarrow \qquad m = A^{-1} \cdot v$$

Nur stellt sich jetzt die Frage, wie wir die Inverse von A berechnen. Dazu können wir wiederum den Ansatz der Fouriertransformation uns zur Hilfe nehmen, jedoch betrachten wir jetzt deren Inverse. Definiert ist sie als

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \mathrm{e}^{-j\omega t} dt \qquad \Rightarrow \qquad \mathfrak{F}^{-1}(F(\omega)) = f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \mathrm{e}^{j\omega t} d\omega.$$

Im wesentlichen ändert sich bei der inversen diskreten Fouriertransformation $e^{j/2\pi}$ zu $e^{-j/2\pi}$. Zusätzlich benötigt die inverse noch einen Korrekturfaktor $1/n$. Wir erwarten daher, dass wir auch im endlichen Körper A die Zahl a durch a^{-1} ersetzen können. Mit der primitiven Einheitswurzel ergibt das

$$8^1 \qquad \rightarrow \qquad 8^{-1}.$$

Mit einem solchen Problem haben wir uns bereits in Abschnitt **??** befasst und so den euklidischen Algorithmus kennengelernt, den wir auf diesen Fall anwenden können.

1.7.1 Inverse der primitiven Einheitswurzel

Die Funktionsweise des euklidischen Algorithmus ist im Abschnitt **??** ausführlich beschrieben. Für unsere Anwendung wählen wir die Parameter $a = 8$ und $b = 11$ ($\mathbb{F}_{11}$). Daraus erhalten wir

k	a_i	b_i	q_i	c_i	d_i
				1	0
0	8	11	0	0	1
1	11	8	1	1	0
2	8	3	2	−1	1
3	3	2	1	3	−2
4	2	1	2	−4	3
5	1	0		11	−8

$$\begin{aligned} -4 \cdot 8 + 3 \cdot 11 &= 1 \\ 7 \cdot 8 + 3 \cdot 11 &= 1 \\ 8^{-1} &= 7 \end{aligned}$$

als Inverse der primitiven Einheitswurzel. Alternativ können wir das Resultat auch aus der Tabelle 1.12.2 ablesen. Die inverse Transformationsmatrix A^{-1} bilden wir, indem wir jetzt die inverse primitive Einheitswurzel anstelle der primitiven Einheitswurzel in die Matrix einsetzen:

$$\begin{pmatrix} 8^0 & 8^0 & 8^0 & 8^0 & \dots & 8^0 \\ 8^0 & 8^{-1} & 8^{-2} & 8^{-3} & \dots & 8^{-9} \\ 8^0 & 8^{-2} & 8^{-4} & 8^{-6} & \dots & 8^{-18} \\ 8^0 & 8^{-3} & 8^{-6} & 8^{-9} & \dots & 8^{-27} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 8^0 & 8^{-9} & 8^{-18} & 8^{-27} & \dots & 8^{-81} \end{pmatrix} \Rightarrow \begin{pmatrix} 7^0 & 7^0 & 7^0 & 7^0 & \dots & 7^0 \\ 7^0 & 7^1 & 7^2 & 7^3 & \dots & 7^9 \\ 7^0 & 7^2 & 7^4 & 7^6 & \dots & 7^{18} \\ 7^0 & 7^3 & 7^6 & 7^9 & \dots & 7^{27} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 7^0 & 7^9 & 7^{18} & 7^{27} & \dots & 7^{81} \end{pmatrix}$$

1.7.2 Der Faktor s

Die diskrete Fouriertransformation benötigt für die Inverse einen Vorfaktor von $\frac{1}{2\pi}$. Wir müssen also damit rechnen, dass wir für die Inverse Transformationsmatrix ebenfalls einen solchen Vorfaktor benötigen. Nur stellt sich jetzt die Frage, wie wir diesen Vorfaktor in unserem Fall ermitteln können. Dafür betrachten wir eine Regel aus der linearen Algebra, nämlich dass

$$A \cdot A^{-1} = E$$

entsprechen muss. Ist dies nicht der Fall, so benötigt A^{-1} eben genau diesen Korrekturfaktor und ändert die Gleichung so zu

$$A \cdot s \cdot A^{-1} = E. \tag{1.6}$$

Somit sollte es für uns ein leichtes Spiel sein, s für unser Beispiel zu ermitteln:

$$\begin{pmatrix} 8^0 & 8^0 & 8^0 & \dots & 8^0 \\ 8^0 & 8^1 & 8^2 & \dots & 8^9 \\ 8^0 & 8^2 & 8^4 & \dots & 8^{18} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 8^0 & 8^9 & 8^{18} & \dots & 8^{81} \end{pmatrix} \cdot \begin{pmatrix} 7^0 & 7^0 & 7^0 & \dots & 7^0 \\ 7^0 & 7^1 & 7^2 & \dots & 7^9 \\ 7^0 & 7^2 & 7^4 & \dots & 7^{18} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 7^0 & 7^9 & 7^{18} & \dots & 7^{81} \end{pmatrix} = \begin{pmatrix} 10 & 0 & 0 & \dots & 0 \\ 0 & 10 & 0 & \dots & 0 \\ 0 & 0 & 10 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 10 \end{pmatrix}$$

Aus der letzten Matrix folgt, dass wir

$$s = \frac{1}{10}$$

als unseren Vorfaktor setzen müssen um, die Gleichung 1.6 zu erfüllen. Da wir in $\mathbb{F}_{11}$ nur mit ganzen Zahlen arbeiten, schreiben wir $\frac{1}{10}$ in 10^{-1} um und bestimmen diese Inverse erneut mit dem euklidischen Algorithmus. So erhalten wir $10^{-1} = 10$ als Vorfaktor in $\mathbb{F}_{11}$.

1.7.3 Allgemeine Decodierung

Wir haben jetzt alles für eine erfolgreiche Rücktransformation vom empfangenen Nachrichtenvektor beisammen. Die allgemeine Gleichung für die Rücktransformation lautet

$$m = s \cdot A^{-1} \cdot v.$$

Setzen wir nun die Werte ein in

$$m = 10 \cdot A^{-1} \cdot v \qquad \Rightarrow \qquad m = 10 \cdot \begin{pmatrix} 7^0 & 7^0 & 7^0 & 7^0 & 7^0 & 7^0 & 7^0 & 7^0 & 7^0 & 7^0 \\ 7^0 & 7^1 & 7^2 & 7^3 & 7^4 & 7^5 & 7^6 & 7^7 & 7^8 & 7^9 \\ 7^0 & 7^2 & 7^4 & 7^6 & 7^8 & 7^{10} & 7^{12} & 7^{14} & 7^{16} & 7^{18} \\ 7^0 & 7^3 & 7^6 & 7^9 & 7^{12} & 7^{15} & 7^{18} & 7^{21} & 7^{24} & 7^{27} \\ 7^0 & 7^4 & 7^8 & 7^{12} & 7^{16} & 7^{20} & 7^{24} & 7^{28} & 7^{32} & 7^{36} \\ 7^0 & 7^5 & 7^{10} & 7^{15} & 7^{20} & 7^{25} & 7^{30} & 7^{35} & 7^{40} & 7^{45} \\ 7^0 & 7^6 & 7^{12} & 7^{18} & 7^{24} & 7^{30} & 7^{36} & 7^{42} & 7^{48} & 7^{54} \\ 7^0 & 7^7 & 7^{14} & 7^{21} & 7^{28} & 7^{35} & 7^{42} & 7^{49} & 7^{56} & 7^{63} \\ 7^0 & 7^8 & 7^{16} & 7^{24} & 7^{32} & 7^{40} & 7^{48} & 7^{56} & 7^{64} & 7^{72} \\ 7^0 & 7^9 & 7^{18} & 7^{27} & 7^{36} & 7^{45} & 7^{54} & 7^{63} & 7^{72} & 7^{81} \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 3 \\ 6 \\ 5 \\ 2 \\ 10 \\ 2 \\ 7 \\ 10 \\ 4 \end{pmatrix}$$

und wir erhalten

$$m = [0, 0, 0, 0, 4, 7, 2, 5, 8, 1]$$

als unsere Nachricht zurück.

1.8 Decodierung: Ansatz mit Fehlerkorrektur

Bisher haben wir die Decodierung unter der Bedingung durchgeführt, dass der Übertragungsvektor fehlerlos versendet und empfangen wurde. In der realen Welt müssen wir uns jedoch damit abfinden, dass kein Übertragungskanal garantiert fehlerfrei ist und das wir früher oder später mit Fehlern rechnen müssen. Genau für dieses Problem wurden Fehler korrigierende Codes, wie der Reed-Solomon-Code, entwickelt. In diesem Abschnitt betrachten wir somit die Idee der Fehlerkorrektur und wie wir diese auf unser Beispiel anwenden können.

Der Übertragungskanal im Beispiel weisst jetzt den Fehlervektor

$$u = [0, 0, 0, 3, 0, 0, 0, 0, 2, 0]$$

auf. Senden wir jetzt unser Übertragungsvektor v durch diesen Kanal, addiert sich der Fehlervektor u auf unsere Übertragung und wir erhalten

v		$[5, 3, 6, 5, 2, 10, 2, 7, 10, 4]$
u	$+$	$[0, 0, 0, 3, 0, 0, 0, 0, 2, 0]$
w		$[5, 3, 6, 8, 2, 10, 2, 7, 1, 4]$

als neuen, fehlerbehafteten Übertragungsvektor w auf der Empfängerseite. Als Empfänger wissen wir jedoch nicht, dass der erhaltene Übertragungsvektor jetzt fehlerbehaftet ist und werden dementsprechend den Ansatz aus Abschnitt 1.7 anwenden. Wir stellen jedoch recht schnell fest, dass am decodierten Nachrichtenblock

$$r = [\underbrace{5, 7, 4, 10}_{\text{Syndrom}}, 5, 4, 5, 7, 6, 7]$$

etwas nicht in Ordnung ist, denn die vorderen vier Fehlerkorrekturstellen haben nicht mehr den Wert null. Der Nachrichtenblock weisst jetzt ein *Syndrom* auf, welches anzeigt, dass der Übertragungsvektor fehlerhaft empfangen wurde. Jetzt stellt sich natürlich die Frage, wie wir daraus den ursprünglich gesendeten Nachrichtenvektor zurückerhalten sollen. Laut der Definition über die Funktionsweise eines Reed-Solomon-Codes können wir aus den Fehlerkorrekturstellen ein "Lokatorpolynom" berechnen, welches die Information enthält, welche Stellen innerhalb des empfangenen Übertragungsvektors fehlerhaft sind.

1.8.1 Das Fehlerstellenpolynom $d(X)$

Bevor wir unser Lokatorpolynom berechnen können, müssen wir zuerst eine Möglichkeit finden, die fehlerhaften von den korrekten Stellen im Übertragungsvektor unterscheiden zu können. In einem ersten Versuch berechnen wir die Differenz d des empfangenen und dem gesendeten Übertragungsvektor mit

$$\begin{aligned} m(X) &= 4X^5 + 7X^4 + 2X^3 + 5X^2 + 8X + 1 \\ r(X) &= 5X^9 + 7X^8 + 4X^7 + 10X^6 + 5X^5 + 4X^4 + 5X^3 + 7X^2 + 6X + 7 \\ d(X) &= r(X) - m(X) \end{aligned}$$

und nennen $d(X)$ als unseres Fehlerstellenpolynom. Dieses Polynom soll uns sagen, welche Stellen korrekt und welche fehlerhaft sind.

Durch das verwenden von $m(X)$ stossen wir auf weitere Probleme, da wir den Nachrichtenvektor auf der Empfängerseite nicht kennen (unser Ziel ist es ja genau diesen zu finden). Dieses Problem betrachten wir im Abschnitt 1.8.5 genauer. Um die Überlegungen in den folgenden Abschnitten besser zu verstehen sei $m(X)$ bekannt auf der Empfängerseite.

Setzen wir jetzt unsere Einheitswurzel aus dem Beispiel ein so erhalten wir

i	0	1	2	3	4	5	6	7	8	9
$r(a^i)$	5	3	6	8	2	10	2	7	1	4
$m(a^i)$	5	3	6	5	2	10	2	7	10	4
$d(a^i)$	0	0	0	3	0	0	0	0	2	0

und damit die Information, dass allen Stellen, die nicht Null sind, Fehler enthalten. Aus der Tabelle lesen wir ab, das in unserem Beispiel die Fehler an der Stelle drei und acht zu finden sind.

Für das einfache Bestimmen von Hand mag dies ja noch ausreichen, jedoch können wir mit diesen Stellen nicht das Lokatorpolynom bestimmen, denn dafür bräuchten wir alle Nullstellen, an denen es Fehler gegeben hat (also sozusagen genau das umgekehrte). Um dies zu erreichen wenden wir eine andere Herangehensweise und nehmen uns den Satz von Fermat sowie den kleinsten gemeinsamen Teiler zur Hilfe.

1.8.2 Mit dem grössten gemeinsamen Teiler auf Nullstellenjagd

Zuerst betrachten wir den Satz von Fermat, dessen Funktionsweise wir in Abschnitt **??** kennengelernt haben. Der besagt, dass

$$f(X) = X^{q-1} - 1 = 0$$

gilt für jedes X. Setzen wir das q von unserem Beispiel ein

$$f(X) = X^{10} - 1 = 0 \qquad \text{für } X \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

und stellen dies als Faktorisierung dar. So ergibt sich die Darstellung

$$f(X) = (X - a^0)(X - a^1)(X - a^2)(X - a^3)(X - a^4)(X - a^5)(X - a^6)(X - a^7)(X - a^8)(X - a^9).$$

Zur Überprüfung können wir unsere Einheitswurzel in a einsetzen und werden sehen, dass wir für $f(X) = 0$ erhalten werden.

Wir können jetzt auch $d(X)$ nach der gleichen Überlegung darstellen als

$$d(X) = (X - a^0)(X - a^1)(X - a^2)(X - a^3)(X - a^4)(X - a^5)(X - a^6)(X - a^7)(X - a^8)(X - a^9) \cdot p(x),$$

wobei diese Darstellung nicht mehr alle Nullstellen umfasst wie es noch in $f(X)$ der Fall war. Dies liegt daran, dass wir ja zwei Fehlerstellen (grau markiert) haben, die nicht Null sind. Diese fassen wir zum Restpolynom $p(X)$ zusammen. Wenn wir jetzt den grössten gemeinsamen Teiler von $f(X)$ und $d(X)$ berechnen, so erhalten wir mit

$$\text{ggT}(f(X), d(X)) = (X - a^0)(X - a^1)(X - a^2)(X - a^3)(X - a^4)(X - a^5)(X - a^6)(X - a^7)(X - a^8)(X - a^9)$$

eine Liste von Nullstellen, an denen es keine Fehler gegeben hat. Dies scheint zuerst nicht sehr hilfreich zu sein, da wir für das Lokatorpolynom ja eine Liste der Nullstellen suchen, an denen es Fehler gegeben hat. Aus diesem Grund berechnen wir im nächsten Schritt das kleinste gemeinsame Vielfache von $f(X)$ und $d(X)$.

1.8.3 Mit dem kgV fehlerhafte Nullstellen finden

Das kgV hat nämlich die Eigenschaft sämtliche Nullstellen zu finden, also nicht nur die fehlerhaften sondern auch die korrekten, was in

$$\text{kgV}(f(X), d(X)) = (X-a^0)(X-a^1)(X-a^2)(X-a^3)(X-a^4)(X-a^5)(X-a^6)(X-a^7)(X-a^8)(X-a^9)\cdot q(X).$$

ersichtlich ist. Aus dem vorherigen Abschnitt wissen wir auch, dass $d(X)$ alle korrekten Nullstellen beinhaltet. Teilen wir das kgV jetzt auf in

$$\text{kgV}(f(X), d(X)) = d(X) \cdot l(X),$$

sollten wir für $l(X)$ eine Liste mit allen fehlerhaften Nullstellen erhalten. Somit ist

$$l(X) = (X - a^3)(X - a^8)$$

unser gesuchtes Lokatorpolynom. Es scheint so als müssten wir nur noch an den besagten Stellen den Übertragungsvektor korrigieren und wir währen fertig mit der Fehlerkorrektur. Jedoch haben wir noch ein grundlegendes Problem, dass zu Beginn aufgetaucht ist, wir aber beiseite geschoben haben. Die Rede ist natürlich vom Nachrichtenvektor $m(X)$, mit dem wir in erster Linie das wichtige Fehlerstellenpolynom $d(X)$ berechnet haben, auf der Empfängerseite aber nicht kennen.

1.8.4 Der problematische Nachrichtenvektor $m(X)$

In Abschnitt 1.8 haben wir

$$d(X) = r(X) - m(X)$$

in Abhängigkeit von $m(X)$ berechnet. Jedoch haben wir ausser acht gelassen, dass $m(X)$ auf der Empfängerseite nicht verfügbar und somit gänzlich unbekannt ist. Es scheint so als würde dieser Lösungsansatz, den wir bisher verfolgt haben, nicht funktioniert. Wir könnten uns höchstens noch fragen, ob wir tatsächlich nichts über den Nachrichtenvektor im Beispiel wissen.

Wenn wir noch einmal den Vektor betrachten als

$$m = [0, 0, 0, 0, 4, 7, 2, 5, 8, 1]$$

fällt uns aber auf, dass wir doch etwas über diesen Vektor wissen, nämlich den Wert der ersten $2t$ (im Beispiel vier) Stellen. Im Normalfall sollen diese nämlich den Wert 0 haben und somit sind nur die letzten k Stellen (im Beispiel sechs) für uns unbekannt, dargestellt als

$$m = [0, 0, 0, 0, ?, ?, ?, ?, ?, ?].$$

Nach der Definition des Reed-Solomon-Codes soll an genau diesen vier Stellen auch die Information befinden, wo die Fehlerstellen liegen. Daher reicht es auch aus

$$d(X) = 5X^9 + 7X^8 + 4X^7 + 10X^6 + p(X)$$

so zu berechnen, dass wir die wichtigen vier Stellen kennen, der Rest des Polynoms jedoch im unbekannten Restpolynom $p(X)$ enthalten ist.

1.8.5 Die Berechnung der Fehlerstellen

Um die Fehlerstellen zu berechnen wenden wir die gleiche Vorgehensweise wie zuvor an, also zuerst den ggT, danach berechnen wir das kgV um am Ende das Lokatorpolynom zu erhalten.

Schritt 1: ggT

Wir berechnen den ggT von $f(X)$ und $d(X)$ mit

$$\begin{aligned} f(X) &= X^{10} - 1 = X^{10} + 10 \\ d(X) &= 5X^9 + 7X^8 + 4X^7 + 10X^6 + p(X) \end{aligned}$$

$$\begin{array}{rl} X^{10} + 10 & : 5X^9 + 7X^8 + 4X^7 + 10X^6 + p(X) = 9X + 5 \\ \underline{X^{10} + 8X^9 + 3X^8 + 2X^7 + p(X)} & \\ 3X^9 + 8X^8 + 9X^7 + p(X) & \\ \underline{3X^9 + 2X^8 + 9X^7 + p(X)} & \\ 6X^8 + 0X^7 + p(X) & \end{array}$$

$$\begin{array}{rll} 5X^9 + 7X^8 + 4X^7 + 10X^6 + p(X) & : 6X^8 + 0X^7 & = 10X + 3 \\ \underline{5X^9 + 0X^8 + p(X)} & & \\ 7X^8 + p(X) & & \end{array}$$

und erhalten

$$\mathrm{ggT}(f(X), e(X)) = 6X^8.$$

Schritt 2: kgV

Mit dem Resultat das wir vom ggT erhalten haben können wir jetzt das kgV berechnen. Dazu können wir jetzt den erweiterten Euklidischen Algorithmus verwenden, den wir in Abschnitt **??** kennengelernt haben.

k	q_i	e_i	f_i
		0	1
0	$9X+5$	1	0
1	$10X+3$	$9X+5$	1
2		$2X^2+0X+5$	$10X+3$

Daraus erhalten wir die Faktoren

$$l(X) = 2X^2 + 5 \qquad \rightarrow \qquad l(X) = 2(X-5)(X-6).$$

Schritt 3: Fehlerstellen bestimmen

Unser gesuchtes Lokatorpolynom hat also die Form

$$l(X) = (X - a^i)(X - a^j).$$

Also brauchen wir nur noch i und j zu berechnen und wir haben unsere gesuchten Fehlerstellen. Diese bekommen wir recht einfach mit

$$\begin{aligned} a^i = 5 \qquad &\Rightarrow \qquad i = 3 \\ a^j = 6 \qquad &\Rightarrow \qquad j = 8. \end{aligned}$$

Schlussendlich erhalten wir

$$d(X) = (X - a^3)(X - a^8)$$

als unser Lokatorpolynom mit den fehlerhaften Stellen.

1.9 Nachricht rekonstruieren

Im letzten Abschnitt haben wir eine Möglichkeit gefunden, wie wir die fehlerhaften Stellen lokalisieren können. Mit diesen Stellen soll es uns nun möglich sein, aus dem fehlerhaften empfangenen Nachrichtenvektor wieder unsere Nachricht zu rekonstruieren. Das Lokatorpolynom

$$l(X) = (X - a^3)(X - a^8)$$

markiert dabei diese fehlerhaften Stellen im Übertragungsvektor

$$w = [5, 3, 6, 8, 2, 10, 2, 7, 1, 4].$$

Als Ausgangslage verwenden wir die Matrix, mit der wir den Nachrichtenvektor ursprünglich codiert haben. Unser Ziel ist es wie auch schon im Abschnitt 1.7 eine Möglichkeit zu finden, wie wir den Übertragungsvektor decodieren können. Aufgrund der Fehlerstellen müssen wir aber davon ausgehen, das wir nicht mehr den gleichen Weg verfolgen können wie wir im Abschnitt 1.7 angewendet haben.

Wir stellen also die Matrix auf und markieren gleichzeitig die Fehlerstellen:

$$\begin{pmatrix} a^0 \\ a^1 \\ a^2 \\ a^3 \\ a^4 \\ a^5 \\ a^6 \\ a^7 \\ a^8 \\ a^9 \end{pmatrix} \begin{pmatrix} 5 \\ 3 \\ 6 \\ 8 \\ 2 \\ 10 \\ 2 \\ 7 \\ 1 \\ 4 \end{pmatrix} = \begin{pmatrix} 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 \\ 8^0 & 8^1 & 8^2 & 8^3 & 8^4 & 8^5 & 8^6 & 8^7 & 8^8 & 8^9 \\ 8^0 & 8^2 & 8^4 & 8^6 & 8^8 & 8^{10} & 8^{12} & 8^{14} & 8^{16} & 8^{18} \\ 8^0 & 8^3 & 8^6 & 8^9 & 8^{12} & 8^{15} & 8^{18} & 8^{21} & 8^{24} & 8^{27} \\ 8^0 & 8^4 & 8^8 & 8^{12} & 8^{16} & 8^{20} & 8^{24} & 8^{28} & 8^{32} & 8^{36} \\ 8^0 & 8^5 & 8^{10} & 8^{15} & 8^{20} & 8^{25} & 8^{30} & 8^{35} & 8^{40} & 8^{45} \\ 8^0 & 8^6 & 8^{12} & 8^{18} & 8^{24} & 8^{30} & 8^{36} & 8^{42} & 8^{48} & 8^{54} \\ 8^0 & 8^7 & 8^{14} & 8^{21} & 8^{28} & 8^{35} & 8^{42} & 8^{49} & 8^{56} & 8^{63} \\ 8^0 & 8^8 & 8^{16} & 8^{24} & 8^{32} & 8^{40} & 8^{48} & 8^{56} & 8^{64} & 8^{72} \\ 8^0 & 8^9 & 8^{18} & 8^{27} & 8^{36} & 8^{45} & 8^{54} & 8^{63} & 8^{72} & 8^{81} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \end{pmatrix}.$$

Die rot markierten Stellen im Übertragungsvektor enthalten Fehler und bringt uns daher keinen weiteren Nutzen. Aus diesem Grund werden diese Stellen aus dem Vektor entfernt, was wir hier ohne Probleme machen können, da dieser Code ja über Fehlerkorrekturstellen verfügt, deren Aufgabe es ist, eine bestimmte Anzahl an Fehler kompensieren zu können. Die dazugehörigen Zeilen in der Matrix werden ebenfalls entfernt, da die Matrix gleich viele Zeilen wie im Übertragungsvektor aufweisen muss, damit man ihn decodieren kann.

Daraus resultiert

$$\begin{pmatrix} 5 \\ 3 \\ 6 \\ 2 \\ 10 \\ 2 \\ 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 \\ 8^0 & 8^1 & 8^2 & 8^3 & 8^4 & 8^5 & 8^6 & 8^7 & 8^8 & 8^9 \\ 8^0 & 8^2 & 8^4 & 8^6 & 8^8 & 8^{10} & 8^{12} & 8^{14} & 8^{16} & 8^{18} \\ 8^0 & 8^4 & 8^8 & 8^{12} & 8^{16} & 8^{20} & 8^{24} & 8^{28} & 8^{32} & 8^{36} \\ 8^0 & 8^5 & 8^{10} & 8^{15} & 8^{20} & 8^{25} & 8^{30} & 8^{35} & 8^{40} & 8^{45} \\ 8^0 & 8^6 & 8^{12} & 8^{18} & 8^{24} & 8^{30} & 8^{36} & 8^{42} & 8^{48} & 8^{54} \\ 8^0 & 8^7 & 8^{14} & 8^{21} & 8^{28} & 8^{35} & 8^{42} & 8^{49} & 8^{56} & 8^{63} \\ 8^0 & 8^9 & 8^{18} & 8^{27} & 8^{36} & 8^{45} & 8^{54} & 8^{63} & 8^{72} & 8^{81} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \end{pmatrix}.$$

Die Matrix ist jedoch nicht mehr quadratisch, was eine Rekonstruktion durch Inversion ausschliesst. Um die quadratische Form wieder herzustellen müssen wir zwei Spalten aus der Matrix entfernen. Wir kennen aber das Resultat aus den letzten vier Spalten, da wir wissen, das die Nachricht aus Nutzdatenteil und Fehlerkorrekturteil besteht, wobei der letzteres bekanntlich aus lauter Nullstellen besteht. Wir nehmen die markierten Spalten in

$$\begin{pmatrix} 5 \\ 3 \\ 6 \\ 2 \\ 10 \\ 2 \\ 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 \\ 8^0 & 8^1 & 8^2 & 8^3 & 8^4 & 8^5 & 8^6 & 8^7 & 8^8 & 8^9 \\ 8^0 & 8^2 & 8^4 & 8^6 & 8^8 & 8^{10} & 8^{12} & 8^{14} & 8^{16} & 8^{18} \\ 8^0 & 8^4 & 8^8 & 8^{12} & 8^{16} & 8^{20} & 8^{24} & 8^{28} & 8^{32} & 8^{36} \\ 8^0 & 8^5 & 8^{10} & 8^{15} & 8^{20} & 8^{25} & 8^{30} & 8^{35} & 8^{40} & 8^{45} \\ 8^0 & 8^6 & 8^{12} & 8^{18} & 8^{24} & 8^{30} & 8^{36} & 8^{42} & 8^{48} & 8^{54} \\ 8^0 & 8^7 & 8^{14} & 8^{21} & 8^{28} & 8^{35} & 8^{42} & 8^{49} & 8^{56} & 8^{63} \\ 8^0 & 8^9 & 8^{18} & 8^{27} & 8^{36} & 8^{45} & 8^{54} & 8^{63} & 8^{72} & 8^{81} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \end{pmatrix}.$$

aus der Matrix heraus und erhalten so das Überbestimmte Gleichungssystem

$$\begin{pmatrix} 5 \\ 3 \\ 6 \\ 2 \\ 10 \\ 2 \\ 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 \\ 8^0 & 8^1 & 8^2 & 8^3 & 8^4 & 8^5 \\ 8^0 & 8^2 & 8^4 & 8^6 & 8^8 & 8^{10} \\ 8^0 & 8^4 & 8^8 & 8^{12} & 8^{16} & 8^{20} \\ 8^0 & 8^5 & 8^{10} & 8^{15} & 8^{20} & 8^{25} \\ 8^0 & 8^6 & 8^{12} & 8^{18} & 8^{24} & 8^{30} \\ 8^0 & 8^7 & 8^{14} & 8^{21} & 8^{28} & 8^{35} \\ 8^0 & 8^9 & 8^{18} & 8^{27} & 8^{36} & 8^{45} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{pmatrix}.$$

Die roten Zeilen können wir aufgrund der Überbestimmtheit ebenfalls entfernen und erhalten so die gesuchte quadratische Matrix

$$\begin{pmatrix} 5 \\ 3 \\ 6 \\ 2 \\ 10 \\ 2 \end{pmatrix} = \begin{pmatrix} 8^0 & 8^0 & 8^0 & 8^0 & 8^0 & 8^0 \\ 8^0 & 8^1 & 8^2 & 8^3 & 8^4 & 8^5 \\ 8^0 & 8^2 & 8^4 & 8^6 & 8^8 & 8^{10} \\ 8^0 & 8^4 & 8^8 & 8^{12} & 8^{16} & 8^{20} \\ 8^0 & 8^5 & 8^{10} & 8^{15} & 8^{20} & 8^{25} \\ 8^0 & 8^6 & 8^{12} & 8^{18} & 8^{24} & 8^{30} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{pmatrix}.$$

Nun können wir den Gauss-Algorithmus anwenden um die Matrix zu Invertieren.

$$\begin{pmatrix} 5 \\ 3 \\ 6 \\ 2 \\ 10 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 8 & 9 & 6 & 4 & 10 \\ 1 & 9 & 4 & 3 & 5 & 1 \\ 1 & 4 & 5 & 9 & 3 & 1 \\ 1 & 10 & 1 & 10 & 1 & 10 \\ 1 & 3 & 9 & 5 & 4 & 1 \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{pmatrix} = \begin{pmatrix} 6 & 4 & 4 & 6 & 2 & 1 \\ 2 & 7 & 10 & 3 & 4 & 7 \\ 1 & 8 & 9 & 8 & 3 & 4 \\ 3 & 6 & 6 & 4 & 5 & 9 \\ 10 & 10 & 9 & 8 & 1 & 6 \\ 1 & 9 & 6 & 4 & 7 & 6 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 3 \\ 6 \\ 2 \\ 10 \\ 2 \end{pmatrix}$$

Multiplizieren wir nun aus, erhalten wir unseren Nutzdatenteil

$$m = [4, 7, 2, 5, 8, 1]$$

zurück, den wir ursprünglich versendet haben.

Wir möchten noch anmerken, dass es mehrere Wege für die Rekonstruktion des Nutzdatenteils gibt, diese aber alle auf dem Lokatorpolynom basieren.

1.10 Zusammenfassung

Dieser Abschnitt beinhaltet eine Übersicht über die Funktionsweise eines Reed-Solomon-Codes für beliebige endliche Körper.

Schritt 1: primitives Element

Zu Beginn soll entschieden werden, in welchem endlichen Körper $\mathbb{F}_q$ gerechnet werden soll. Ausserdem muss im gewählten Körper eine primitive Einheitswurzel gefunden, bzw. bestimmt werden.

Schritt 2: Codierung

Für die Codierung wird die Nachricht als Koeffizienten des Polynoms $m(X)$ geschrieben, anschliessend wird a^i in $m(X)$ eingesetzt. Daraus ergibt sich die Codierungsmatrix

$$A(a) = \begin{pmatrix} a^0 & a^0 & a^0 & \dots \\ a^0 & a^1 & a^2 & \dots \\ a^0 & a^2 & a^4 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Mit dieser Matrix können wir den Nachrichtenblock zum Übertragungsvektor codieren.

Schritt 3: Decodierung ohne Fehler

Im ersten Schritt zur Decodierung muss geprüft werden, ob der Übertragungsvektor Fehler beinhaltet. Ist dies nicht der Fall, so kann die Matrix $A(a)$ invertiert werden mit

$$A(a)^{-1} = \frac{1}{q-1} \cdot A(a^{-1}).$$

Die Codierungsmatrix ändert sich somit zur Decodierungsmatrix

$$\begin{pmatrix} a^0 & a^0 & a^0 & \dots \\ a^0 & a^1 & a^2 & \dots \\ a^0 & a^2 & a^4 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} = \frac{1}{q-1} \cdot \begin{pmatrix} a^0 & a^0 & a^0 & \dots \\ a^0 & a^{-1} & a^{-2} & \dots \\ a^0 & a^{-2} & a^{-4} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Daraus lässt sich der Nachrichtenblock aus dem Übertragungsvektor rekonstruieren.

Schritt 4: Decodierung mit Fehler

Sollte der Übertragungsvektor fehlerhaft empfangen werden, so kann der Nachrichtenblock nicht durch invertieren der Matrix rekonstruiert werden. Zur Lokalisierung der Fehlerstellen nehmen wir das Polynom $f(X)$ zur Hilfe, welches wir über den Satz von Fermat bestimmt haben. Berechnen wir daraus das kgV von $f(X)$ und $d(X)$, so erhalten wir ein Lokatorpolynom. Durch das bestimmen der Exponenten erhalten wir die Fehlerhaften Stellen im Übertragungsvektor. Für die Rekonstruktion stellen wir ein Gleichungssystem auf und entfernen daraus die Fehlerhaften Zeilen. Im Anschluss kann das verkleinerte Gleichungssystem gelöst werden. Als Resultat erhalten wir die fehlerfreie Nachricht.

1.11 Anwendungen des Reed-Solomon-Codes

In den vorherigen Abschnitten haben wir betrachtet, wie Reed-Solomon-Codes in der Theorie funktionieren. In diesem Abschnitt werden wir einige Anwendungen vorstellen, bei denen ein Reed-Solomon-Code zum Einsatz kommt.

Dabei teilen all diese Anwendungen das gleiche Problem: Die Daten können nur durch einen (höchst Wahrscheinlichen) fehlerbehafteten Kanal empfangen werden. Es gibt keine andere Methode, an diese Daten zu kommen, als über diesen Kanal.

In der Netzwerktechnik zum Beispiel ist es üblich, dass bei Paketverluste oder beschädigt empfangene Datenpaketen diese einfach noch einmal innert wenigen Millisekunden angefordert werden können. In der Raumfahrt ist dies nicht möglich, da aufgrund der beschränkten Speichermöglichkeit die gesammelten Daten so rasch wie möglich zur Erde gesendet werden. Diese Daten wiederum brauchen aufgrund der grossen Distanz Stunden bis die Daten beim Empfänger ankommen. Fehlerhafte Daten kann also auf Grund der Zeitverzögerung nicht mehr angefordert werden.

Bei CDs oder DVDs gibt es zwar kein zeitliches Problem, jedoch erschweren Kratzer, Verschmutzungen oder Produktionsfehler das Lesen einer solchen Disk. Da vor allem Produktionsfehler und Kratzer irreversibel sind und die Disk nicht nach jedem Kratzer ersetzt werden muss, so wird die korrekte Ausgabe der gespeicherten Information durch die Fehlerkorrektur sichergestellt.

Einen ähnlichen Ansatz verfolgen QR-Codes, wobei die Information auch dann noch gelesen werden kann wenn der Code nicht mehr vollständig vorhanden ist.

Obwohl alle diese Codes nach dem gleichen Prinzip arbeiten gibt es starke Unterschiede in deren Funktionsweise. Dies kommt vor allem daher, da die Codes nur Ressourcen zur Verfügung haben, die von der Hardware bereitstellt wird, auf denen die Codes implementiert wurden. Diese Codes bedienen sich daher verschiedener Tricks und Optimierungen um möglichst effizient zu arbeiten.

Um die Fähigkeit eines verwendeten Reed-Solomon-Codes zu beschreiben verwendet man die Notation (n,k), wobei n die Grösse des Nachrichtenblocks angibt und k die Anzahl der Stellen, die für Nutzdaten gebraucht werden können.

1.11.1 Raumfahrt

Obwohl Reed-Solomon-Codes bereits in den 1960er entwickelt wurden fanden sie erstmals Anwendung in der Voyager Raumsonde der NASA. Die Daten der zwei im Jahre 1977 gestarteten Sonden (siehe Abbildung 1.3) werden mit einem (255,233)-Code Codiert. Der Nachrichtenblock hat somit eine Länge von 255 Zahlen, wovon 233 als Nutzlast zur Verfügung stehen. Damit ist es möglich bis zu 11 Fehler im Nachrichtenblock zu korrigieren. Der Codierte Nachrichtenblock wird in kleinere Blöcke aufgeteilt, mit einem Faltungscode erneut Codiert und anschliessend gesendet. Ein Faltungscode ist wie ein Reed-Solomon-Code in der Lage Fehler zu korrigieren, Codiert seine Information aber auf eine andere weise. Aus jedem unterteilten Block wird vor dem Versenden ein Paritätsbit erzeugt und dem Block angehängt. Anhand diesem Paritätsbit überprüft der Empfänger, ob bei der Übertragung der Block beschädigt wurde. Ist dies der Fall, wird der Block bei der Decodierung nicht beachtet. Diese so entstandenen “Lücken” im Datenstrom werden wiederum vom Reed-Solomon-Code korrigiert. Dieses Zusammenspiel beider Codes garantiert so eine hohe Robustheit gegenüber Übertragungsfeher.

1.11.2 CD/DVD

Compact discs verwenden sogar zwei ineinander verschachtelte Reed-Solomon-Codes, einen (32,28)-Code und einen (28,24)-Code. Beide Codes sind in der Lage, Fehler aus dem jeweils anderen gelesenen Block zu korrigieren. Dieses spezielle Zusammenspielen dieser beiden Codes werden auch Cross-interleaved Reed-Solomon-Codes (CIRC) genannt. Diese Vorgehensweise erzielt eine hohe Robustheit gegenüber Produktionsfehlern oder Verschmutzung auf der Disc. Bei CDs sind diese in der Lage, bis zu 4000 fehlerhafte Bits am Stück (ca. 2.5mm) zu erkennen und zu korrigieren.

Die Digital Video Disc funktioniert nach dem selben Konzept mit grösseren Codeblöcken. Die DVD verwendet einen (208,192)-Code und einen (182,172)-Code.

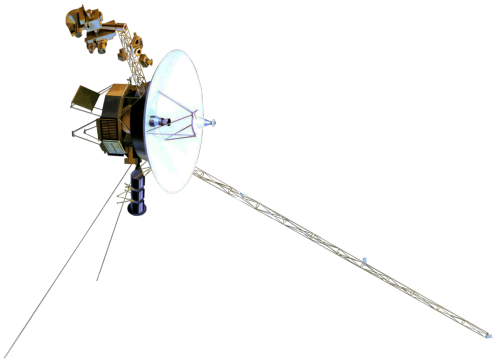
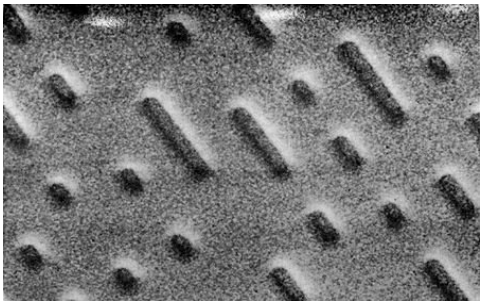


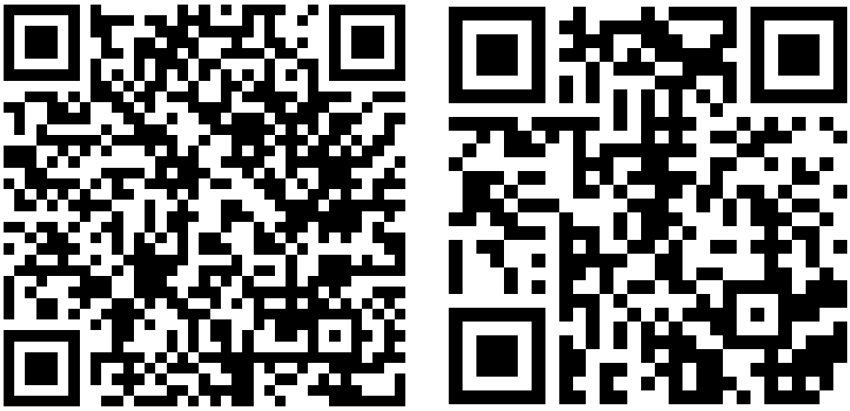
Abbildung 1.3: Mit einer Entfernung von über 22.8 Milliarden Kilometer ist die Voyager 1 Raumsonde das am weitesten entfernte, von Menschen erschaffene Objekt. Obwohl ihre Schwestersonde Voyager 2 zuerst ins All gestartet wurde befindet Sie sich "nur" 19 Milliarden Kilometer weit weg von der Erde. Aufgrund abnehmender Batterieleistung werden die beiden Sonden ihre wissenschaftlichen Aktivitäten etwa 2025 einstellen, bleiben aber bis in die 2030er mit uns in Kontakt.

(a)



(b)

Abbildung 1.4: CDs kamen 1982 auf den Markt. Sie funktioniert durch das Einpressen oder Einbrennen von Punkten und Strichen, die die Daten repräsentieren. Gelesen werden diese wiederum durch die Reflektion eines Lasers an diesen Punkten und Strichen.



(a) (b)

Abbildung 1.5: Anhand der grösse würde man darauf schliessen, dass bei (a) mehr Informationen Codiert sind als bei (b). Tatsächlich aber beinhalten beide Codes die gleiche Information. Das liegt daran, da die Fehlerkorrekturfähigkeit von QR-Codes sich in insgesamt vier Levels aufteilen lassen. Der höchste Fehlerkorrektur-Level, der bei (a) angewendet wurde, ist in der Lage, bis zu 30% der Daten wiederherzustellen. Der kleinste Level schafft etwa 7%, der in (b) veranschaulicht wird. Da die Grösse also nichts über die Menge an Daten aussagt, könnte es sich bei (a) auch um einen Code mit viel Nutzdaten und kleinem Fehlerkorrektur-Level handeln. Der Unterschied ist von Auge nicht sichtbar.

1.11.3 QR-Codes

Quick Response Codes oder auch QR-Codes funktionieren nach einem sehr ähnlichen Prinzip wie in unserem Beispiel der Abschnitte 1.6 - 1.9 nur das QR-Codes in einem $\mathbb{F}_{256}$ Körper arbeiten. Die physische Grösse eines Codes ist stark abhängig von der Menge an codierten Daten sowie dem verwendeten Fehlerkorrektur-Level. Es ist so auf dem ersten Blick nicht ersichtlich, wie viel Nutzinformationen ein Qr-Code enthält. Die QR-Codes in Abbildung 1.5 zeigen jeweils die Gleiche Information mit unterschiedlichem Fehlerkorrektur-Level. Codes mit einem höheren Korrektur-Level können auch für Designer-Codes Zweckentfremdet werden. Dabei wird z.B. das Firmenlogo oder einen Schriftzug über den Qr-Code gelegt, ohne das die Funktion des Codes beeinträchtigt wird. Ein Beispiel dazu ist unter Abbildung 1.6 zu finden.

1.12 Hilfstabellen für $\mathbb{F}_{11}$

Um das rechnen zu erleichtern findet man in diesem Abschnitt die Resultate, die bei der Addition und der Multiplikation in $\mathbb{F}_{11}$ resultieren.

(a) (b)

Abbildung 1.6: Während (a) noch einen unveränderten QR-Code repräsentiert, handelt es sich bei (b) nun um einen Designer-QR-Code. Beide Codes verfügen über einen mittleren Fehlerkorrektur-Level von theoretisch 15%. Da bei (b) jetzt einen Teil des Codes durch ein Logo verdeckt wird, schränkt sich die Fehlerkorrekturfähigkeit je nach Grösse des verdeckten Teils mehr oder weniger stark ein. Unser Designer-Code in (b) ist nur noch in der Lage etwa 9% des Codes zu rekonstruieren.

1.12.1 Additionstabelle

+	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	0
2	2	3	4	5	6	7	8	9	10	0	1
3	3	4	5	6	7	8	9	10	0	1	2
4	4	5	6	7	8	9	10	0	1	2	3
5	5	6	7	8	9	10	0	1	2	3	4
6	6	7	8	9	10	0	1	2	3	4	5
7	7	8	9	10	0	1	2	3	4	5	6
8	8	9	10	0	1	2	3	4	5	6	7
9	9	10	0	1	2	3	4	5	6	7	8
10	10	0	1	2	3	4	5	6	7	8	9

1.12.2 Multiplikationstabelle

·	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	1	3	5	7	9
3	0	3	6	9	1	4	7	10	2	5	8
4	0	4	8	1	5	9	2	6	10	3	7
5	0	5	10	4	9	3	8	2	7	1	6
6	0	6	1	7	2	8	3	9	4	10	5
7	0	7	3	10	6	2	9	5	1	8	4
8	0	8	5	2	10	7	4	1	9	6	3
9	0	9	7	5	3	1	10	8	6	4	2
10	0	10	9	8	7	6	5	4	3	2	1