Einführung in die Krytographie sowie die Informations- und Codierungstheorie

Vorlesungsskript Universität Koblenz, Sommersemester 2015

Carolin Torchiani Michael Helmling

Version vom 27. Oktober 2015

Über dieses Skript

Das vorliegende Vorlesungsskript *Einführung in die Kryptographie sowie die Informations- und Codierungstheorie* wurde von Carolin Torchiani[1] und Michael Helmling[2] für eine vierstündige Vorlesung erstellt, die im Sommersemester 2015 an der Uni Koblenz für Masterstudenten der Mathematik oder Informatik gehalten wurde.

Wir haben dieses Skript auf Basis verschiedener Quellen von Grund auf neu erstellt, da die ohnehin erstaunlich dünne Auswahl vergleichbarer Skripte kein aus unserer Sicht zufriedenstellendes Gesamtkonzept beinhaltete. Um anderen dieselbe Arbeit zu ersparen bzw. zu erleichtern, ist das Skript inklusive der TEX-Quellen[3] frei zugänglich. Sollten Sie es ganz oder in Teilen für eigene Lehrveranstaltungen nutzen, würden wir uns sehr über eine kurze Rückmeldung freuen. Ebenso nehmen wir Hinweise auf Fehler oder inhaltliche Verbesserungsvorschläge jederzeit gerne an.

Neben dem Skript selbst beinhaltet das oben erwähnte Git-Repository eine Sammlung von Übungsblättern für ein vorlesungsbegleitendes zweistündiges Tutorium, einen Einführungsvortrag zur Geschichte der Kryptographie sowie zwei kürzere Foliensätze zu den Themen *AES* und *Verschlüsselungsmodi*. Hinweise zum Übersetzen der TEX-Quellen finden sich in der beiliegenden `README`-Datei.

Carolin Torchiani & Michael Helmling
Koblenz, Oktober 2015

[1] carolin.torchiani@posteo.de
[2] michaelhelmling@posteo.de
[3] siehe `https://github.com/supermihi/KryptoCodierung`

Inhaltsverzeichnis

Teil I.

Kryptographie

Empfohlene Literatur

Kryptographie

Johannes Buchmann. *Einführung in die Kryptographie*. Springer, 1999

Mohamed Barakat und Timo Hanke. *Cryptography — Lecture notes*. 2012. URL: http://www.mathematik.uni-kl.de/~barakat/Lehre/WS10/Cryptography/lecture_notes/Cryptography.pdf

Christian Karpfinger und Hubert Kiechle. *Kryptologie*. Vieweg+Teubner, 2010

Jeffrey Hoffstein, Jill Pipher und Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer, 2008

Algebraische Grundlagen

Andreas Gathmann. *Algebraische Strukturen*. 2010. URL: http://www.mathematik.uni-kl.de/agag/mitglieder/professoren/gathmann/notes/agstr/

Thomas Markwig. *Elementare Zahlentheorie*. 2010. URL: http://www.mathematik.uni-kl.de/~keilen/download/LectureNotes/zahlentheorie.pdf

Harald Scheid und Wolfgang Schwarz. *Elemente der Arithmetik und Algebra*. Spektrum, 2008

Vorbemerkungen

In der klassischen Kryptographie geht es darum, Techniken zu entwickeln, mit denen Nachrichten *vertraulich* ausgetauscht werden können. Nur der Empfänger soll den Inhalt der Nachricht entschlüsseln können, nicht aber ein Lauscher, der die Nachricht während der Zustellung abgefangen hat.

Bereits im dritten Jahrtausend v. Chr. wurden kryptographische Verfahren eingesetzt, das älteste bekannte Buch über die Entschlüsselung kryptographischer Nachrichten stammt aus dem 9. Jahrhundert. Kryptographische Verfahren wurden ursprünglich hauptsächlich vom Militär und in der Politik eingesetzt. Spätestens seit der flächendeckenden Verbreitung des Internets spielen kryptographische Verfahren, die die Vertraulichkeit von Nachrichten schützen, auch im Alltag eine bedeutende Rolle, z. B. beim Eingeben von Passwörtern.

Zusätzlich zur *Vertraulichkeit* hat die moderne Kryptographie weitere Hauptziele, um Informationen zu schützen:

- *Integrität:* Der Empfänger einer Nachricht soll nachprüfen können, ob die Nachricht während der Zustellung verändert wurde. Wichtig ist das zum Beispiel bei Überweisungen, bei denen der Betrag oder der Empfänger nicht nachträglich geändert werden sollen, und bei Computerprogrammen, durch deren Installation man sich keine Viren einfangen möchte.

- *Authentizität:* Der Absender einer Nachricht soll eindeutig identifizierbar sein, so dass der Empfänger sicher sein kann, von wem die Nachricht geschickt wurde. Beispielsweise sollen beim Online-Banking nur berechtigte Personen auf die Kontodaten zugreifen können; beim Online-Shopping will der Anbieter wissen, welcher seiner Kunden ein Produkt geordert hat.

- *Zurechenbarkeit:* Gegenüber Dritten soll es beweisbar sein, dass eine Nachricht von einem bestimmten Empfänger stammt. Auf Papierdokumenten übernimmt eine gewöhnliche Unterschrift diese Funktion; bei elektronischen Nachrichten werden dazu *digitale Signaturen* eingesetzt. Beispielsweise soll es gegenüber einem Richter beweisbar sein, dass ein Vertrag über das Internet geschlossen wurde und wer die Vertragsparteien sind.

In der Vorlesung beschäftigen wir uns damit, wie diese Ziele mittels mathematischer Methoden erreicht werden können. Insbesondere werden kryptographische Verfahren diskutiert, die aktuell im Einsatz sind.

Dieser Teil des Vorlesungsskripts zur Kryptographie basiert hauptsächlich auf „Cryptography — Lecture notes“ von Mohamed Barakat und Timo Hanke [BH12] und auf „Einführung in die Kryptographie“ von Johannes Buchmann [Buc99] . Zum Nachlesen algebraischer Grundlagen ist das Vorlesungsskript „Algebraische Strukturen“ von Andreas Gathmann [Gat10] empfehlenswert.

1. Kryptographische Konzepte

In der klassischen Kryptographie werden Verschlüsselungsverfahren entwickelt, die der vertraulichen Übertragung von Nachrichten dienen. Wir führen Grundbegriffe ein, die die Beschreibung solcher Verfahren ermöglichen, und beschäftigen uns mit Sicherheitseigenschaften und Angriffsarten.

1.1. Kryptosysteme

Notation 1.1 1. Die Menge der natürlichen Zahlen bezeichnen wir mit $\mathbb{N} = \{1, 2, 3, 4, \dots\}$ und setzen $\mathbb{N}_0 = \{0, 1, 2, 3, \dots\}$.

2. Sei $n \in \mathbb{N}$. Wir bezeichnen mit $\mathbb{Z}_n = \{[0]_n, \dots, [n-1]_n\}$ oder einfach $\mathbb{Z}_n = \{[0], \dots, [n]\}$ die Restklassen modulo n.

3. Mit $\mathbb{P} \subset \mathbb{N}$ bezeichnen wir die Menge der Primzahlen. ◁

Zur Erinnerung:

- Für $n \in \mathbb{N}$, $a \in \mathbb{Z}$ ist $[a]_n \in \mathbb{Z}_n$ gegeben durch

$$[a]_n = \{a + kn : k \in \mathbb{Z}\} = \{\dots, a - 3n, a - 2n, a - n, a, a + n, a + 2n, a + 3n, \dots\}.$$

- $(\mathbb{Z}_n, +, \cdot)$ ist ein kommutativer Ring mit Eins und $(\mathbb{Z}_n^m, +)$ ist eine abelsche Gruppe, wobei „+“ für die komponentenweise Addition steht. Beispielsweise gilt

$$[5] + [4] = [9] = [2] \in \mathbb{Z}_7, \quad [5] \cdot [4] = [20] = [6] \in \mathbb{Z}_7$$

und

$$([2], [5]) + ([6], [7]) = ([8], [12]) = ([8], [3]) \in \mathbb{Z}_9^2.$$

Definition 1.2 (Alphabete und Wörter) Ein *Alphabet A* ist eine endliche nichtleere Menge. Ihre Mächtigkeit $|A|$ wird *Länge des Alphabets* genannt und ihre Elemente heißen *Buchstaben.*

1. Ein Element $w = (w_1, \dots, w_n) \in A^n$ heißt *Wort auf A der Länge* $l(w) = n \in \mathbb{N}$. Wir schreiben auch $w = w_1 \dots w_n$.

2. Wir definieren $A^* = \bigcup_{n \in \mathbb{N}_0} A^n$ mit $A^0 = \{\varepsilon\}$, wobei ε ein nicht im Alphabet enthaltenes Symbol ist, das für das *leere Wort* der Länge 0 steht.

3. Seien $v = v_1 \dots v_{n_v}, w = w_1 \dots w_{n_w}$ Wörter auf A. Dann bezeichnet $v \circ w$ die *Verkettung von v und w*, die durch $v \circ w = v_1 \dots v_{n_v} w_1 \dots w_{n_w} \in A^*$ definiert ist.
($\circ : A^* \times A^* \to A^*$ ist also eine zweistellige Operation auf A^*.) ◁

Beispiel 1.3 1. $A = \{\mathtt{a}, \dots \mathtt{z}\}$, $\mathtt{kryptographie} \in A^*$

2. $A = \{0, 1\}$, $101010111 \in A^*$, $1010 \circ 01 = 101001 \in A^*$

3. $A = \{0, \dots, 9\}$, $1239402938 \in A^*$ ◁

Bemerkung 1.4 Das Tupel $(A^*, \circ)$ ist eine Halbgruppe mit neutralem Element ε. Sie ist abelsch genau dann, wenn $|A| = 1$. Außerdem gilt $l(v, w) = l(v) + l(w)$, d. h. $l : (A^*, \circ) \to (\mathbb{Z}_{\geq 0}, +)$ ist ein Halbgruppenhomomorphismus. ◁

Definition 1.5 (Kryptosystem) Ein *Kryptosystem* ist ein 5-Tupel $(\mathcal{P} \subset A_1^*, \mathcal{C} \subset A_2^*, \mathcal{K}, \mathcal{E}, \mathcal{D})$ mit folgenden Eigenschaften:

1. A_1 und A_2 sind Alphabete, das *Klartextalphabet* und das *Chiffretextalphabet*. $\mathcal{P}$ heißt *Klartextraum* und $\mathcal{C}$ *Chiffretextraum*.
2. $\mathcal{K}$ ist eine Menge. Sie heißt *Schlüsselraum*.
3. $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $E_k : \mathcal{P} \to \mathcal{C}$, den *Verschlüsselungsfunktionen*.
4. $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $D_k : \mathcal{C} \to \mathcal{P}$, den *Entschlüsselungsfunktionen*.
5. Für jeden *Verschlüsselungsschlüssel* $e \in \mathcal{K}$ gibt es einen *Entschlüsselungsschlüssel* $d \in \mathcal{K}$, so dass für alle $p \in \mathcal{P}$ die Gleichung
$$D_d(E_e(p)) = p$$
erfüllt ist. ◁

Häufig wählen wir $A_1 = A_2 =: A$ und $\mathcal{P} := \mathcal{C} := A^*$, d. h. Klartext- und Chiffretextraum stimmen überein und sind die gesamte Menge der Wörter auf A.

Beispiel 1.6 (Caesar-Chiffre) Wir modellieren die Caesar-Chiffre als Kryptosystem. Als Klartext- und Chiffretextalphabet wählen wir $A = \mathbb{Z}_{26} = \{[0], \dots, [25]\}$ und als Klartext- und Chiffretextraum $\mathcal{P} = \mathcal{C} = A^*$. Die Elemente von $\mathbb{Z}_{26}$ werden dabei mit den Buchstaben im Standardalphabet $\{A, \dots, Z\}$ identifiziert, so dass der Buchstabe A der Restklasse $[0]$ und der Buchstabe Z der Restklasse $[25]$ entspricht.

Der Schlüsselraum der Caesar-Chiffre ist ebenfalls durch $\mathcal{K} = \mathbb{Z}_{26}$ gegeben. Die zu $k \in \mathbb{Z}_{26}$ gehörige Verschlüsselungsfunktion $E_k : \mathcal{P} \to \mathcal{C}$ arbeitet buchstabenweise und verschiebt jeden Buchstaben eines Wortes $w = w_1 \dots w_n$ um k Stellen:
$$E_k(w) = (w_1 + k) \dots (w_n + k)$$

Die Entschlüsselungsfunktion $D_k : \mathscr{C} \to \mathscr{P}$ verschiebt jeden Buchstaben eines Wortes um $-k$ Stellen:

$$D_k(w) = (w_1 - k) \dots (w_n - k)$$

Der zum Verschlüsselungsschlüssel $k \in \mathbb{Z}_{26}$ gehörige Entschlüsselungsschlüssel ist $-k \in \mathbb{Z}_{26}$.

Das Wort BRUTUS im Standardalphabet entspricht dem Wort $w = [1]\,[17]\,[20]\,[19]\,[20]\,[18]$ im Klartextalphabet. Mit der zu $[10] \in \mathscr{K}$ gehörigen Verschlüsselungsfunktion $E_{[10]}$ wird w abgebildet auf

$$E_{[10]}(w) = [11]\,[27]\,[30]\,[29]\,[30]\,[28] = [11]\,[1]\,[4]\,[3]\,[4]\,[2].$$

Dieses Wort im Chiffretextraum entspricht dem Wort LADCDB auf dem Standardalphabet. ◁

Aufgabe 1.7 Modellieren Sie die Vigenère-Verschlüsselung als Kryptosystem. Identifizieren Sie dazu die Buchstaben des Standardalphabets mit $\mathbb{Z}_{26}$. ◁

Aufgabe 1.8 Zeigen Sie, dass die Verschlüsselungsfunktionen $E_e : \mathscr{P} \to \mathscr{C}$ eines Kryptosystems immer injektiv sind. ◁

Prinzip 1.9 (Kerckhoffs Prinzip, 1883)
Erste Formulierung: *Die kryptographische Stärke eines Kryptosystems darf nicht auf der Geheimhaltung des Kryptosystems, sondern nur auf der Geheimhaltung des Schlüssels beruhen.*
Zweite Formulierung: *Der Angreifer kennt das Kryptosystem.* ◁

Eine einfache Rechtfertigung dieses Prinzips ist dadurch gegeben, dass es auf der einen Seite sehr schwierig ist einen Verschlüsselungsalgorithmus geheim zu halten (*security by obscurity*), wenn er über längere Zeit von einer großen Anzahl von Personen genutzt wird. Auf der anderen Seite ist es deutlich einfacher einen Schlüssel zwischen Sender und Empfänger auszutauschen, verschiedene Schlüssel für verschiedenen Nachrichten zu benutzen und die Schlüssel im Nachgang wieder zu vernichten. Aus dem gleichen Grund ist davon auszugehen, dass jedwede Schwäche eines öffentlichen Kryptosystems nicht lange geheim bleibt.

Kerckhoffs Prinzip ist heutzutage weitläufig akzeptiert. Der größte Nachteil besteht darin, dass auch ein Widersacher dieselben gründlich getesteten und anerkannten Verschlüsselungsalgorithmen nutzen kann.

Definition 1.10 (Secret-Key- und Public-Key-Kryptosysteme) Wir nennen ein Kryptosystem *symmetrisch* oder *Secret-Key-Kryptosystem (SKC)*, wenn für jeden Verschlüsselungsschlüssel $e \in \mathscr{K}$ die Berechnung eines Entschlüsselungsschlüssels $d \in \mathscr{K}$ *zulässig* ist. Ist die Berechnung der Entschlüsselungsschlüssel *unzulässig*, so wird das Kryptosystem *asymmetrisch* oder *Public-Key-Kryptosystem (PKC)* genannt.

„Zulässig“ bedeutet hier, dass die Berechnung in vertretbarer Zeit erfolgen kann, z. B. im Fall $d = e$. ◁

Bemerkung 1.11 1. Während ein Verschlüsselungsschlüssel e bei einem Public-Key-Kryptosystem öffentlich zugänglich ist (public key = öffentlicher Schlüssel), muss e bei einem Secret-Key-Kryptosystem geheim gehalten werden. Der Entschlüsselungsschlüssel d bleibt in jedem Fall geheim.

2. Typischerweise sind die Algorithmen zur Implementierung von Secret-Key-Kryptosystemen effizienter, weshalb sie für einen Großteil des kryptographischen Nachrichtenaustauschs genutzt werden. Dahingegen werden Public-Key-Systeme genutzt, um die benötigten (im Vergleich zur Nachricht kurzen) Schlüssel geheim auszutauschen. ◁

Definition 1.12 (Sicherheitseigenschaften) Informell gesprochen hat ein Kryptosystem die *Sicherheitseigenschaft*

1. *Einwegeigenschaft*, wenn es unzulässig ist, dass ein Angreifer einen zufälligen Chiffretext entschlüsselt,

2. *Nicht-Unterscheidbarkeit* oder *semantische Sicherheit*, wenn es unzulässig ist, dass ein Angreifer für einen gegebenen Chiffretext herausfindet, zu welchem aus einer gegebenen Menge von Klartexten er gehört,

3. *Nicht-Modifizierbarkeit*, wenn es unzulässig ist, dass ein Angreifer einen gegebenen Chiffretext so verändert, dass der zugehörige Klartext Sinn ergibt und dem ursprünglichen Klartext ähnelt. ◁

Bemerkung 1.13 Man kann zeigen, dass NM $\Rightarrow$ NU $\Rightarrow$ EE gilt. ◁

Definition 1.14 (Angriffsarten) Man unterscheidet folgende *Angriffsarten* auf ein Kryptosystem:

1. *Ciphertext-Only-Angriff* : Der Angreifer kennt nur Chiffretexte.

2. *Known-Plaintext-Angriff* : Der Angreifer kennt Paare, die aus einem Klartext und dem zugehörigen Chiffretext bestehen.

3. *Chosen-Plaintext-Angriff (CPA)*: Der Angreifer kann *einmal* Klartexte wählen und erhält die zugehörigen Chiffretexte. („einmal“ bedeutet, dass der Angreifer seine Auswahl an Klartexten nicht in Abhängigkeit davon ändern kann, welche Chiffretexte er erhält.)

4. *Adaptive-Chosen-Ciphertext-Angriff (CCA2)*: Der Angreifer kann immer wieder Chiffretexte wählen und erhält die zugehörigen Klartexte. („immer wieder“ bedeutet, dass die Auswahl der zu entschlüsselnden Chiffretexte von den bereits erhaltenen Klartexten abhängen darf.) Wenn der Angreifer einen speziellen Chiffretext entschlüsseln möchte, darf er natürlich nicht den zugehörigen Klartext erhalten. Normalerweise zielen diese Angriffe darauf ab, den Entschlüsselungsschlüssel $d \in K$ aufzudecken, nicht einen speziellen Chiffretext zu entschlüsseln. ◁

Bemerkung 1.15 Man kann zeigen, dass CCA2 ≻ CPA ≻ KPA ≻ COA gilt, wobei ≻ „stärker als“ bedeutet (z. B. kann ein Kryptosystem, das durch CPA geknackt werden kann, auch durch CCA2 geknackt werden). ◁

Definition 1.16 (Sicherheitsmodell) Ein Kryptosystem erfüllt ein *Sicherheitsmodell A-B*, wenn es die Sicherheitseigenschaft A unter der Angriffsart B wahrt. ◁

Bemerkung 1.17 Man kann zeigen, dass NM-CCA2 = NU-CCA2 gilt. ◁

Beispiel 1.18 NU-CCA2, d. h. Nicht-Unterscheidbarkeit unter Adaptive-Chosen-Ciphertext-Angriffen, ist das stärkste Sicherheitsmodel, das mithilfe der angegebenen Sicherheitseigenschaften und Angriffsarten konstruiert werden kann. Hier ein Beispiel, das dieses Sicherheitsmodell veranschaulicht. H bezeichne einen Herausforderer und A einen Angreifer.

1. H erzeugt einen geheimen Entschlüsselungsschlüssel $d \in \mathcal{K}$ und wählt einen zugehörigen Verschlüsselungsschlüssel $e \in K$, so dass $D_d(E_e(p)) = p$ für alle Klartexte $p \in \mathcal{P}$ gilt.

2. A erzeugt zwei verschiedene Klartexte $p_0, p_1 \in \mathcal{P}$ und übergibt sie an H.

3. H wählt zufällig ein $i \in \{0, 1\}$ und schickt $c = E_e(p_i)$ zurück an A, mit der Aufforderung i zu erraten.

4. A hat Zugriff auf den Entschlüsselungsalgorithmus D_d (aber nicht auf den geheimen Schlüssel d) und kann beliebige Berechnungen durchführen, aber nicht c entschlüsseln.

5. A rät den Wert von i, basierend auf den durchgeführten Berechnungen.

Das Kryptosystem erfüllt NU-CCA2, wenn die Wahrscheinlichkeit, dass A richtig rät, nicht höher als $\frac{1}{2}$ ist, d. h. A kann nicht unterscheiden, ob der Chiffretext c zum Klartext p_0 oder p_1 gehört, obwohl A Zugriff auf den Entschlüsselungsalgorithmus hat. ◁

Aufgabe 1.19 Zur Enigma. Bestimmen Sie den prozentualen Anteil der Permutationen auf einem Alphabet der Länge 26, die involutorisch und fixpunktfrei sind. ◁

1.2. Perfekte Sicherheit

Im einführenden Vortrag wurden ausschließlich Kryptosysteme beschrieben, die geknackt werden können. Daher stellt sich die Frage, ob es überhaupt „sichere“ Kryptosysteme gibt. Sie wurde 1949 von Claude Shannon positiv beantwortet. Er führte den Begriff der *perfekten Sicherheit* ein und beschrieb ein Kryptosystem, das diese Eigenschaft erfüllt. Um die Theorie von Shannon darzustellen, benötigen wir einige Begriffe und Ergebnisse aus der Wahrscheinlichkeitstheorie.

1.2.1. Kompaktkurs Stochastik I

Alles in diesem Abschnitt sollte Ihnen aus der Stochastik-Vorlesung im Bachelor bereits bekannt sein. Da wir nur endliche Verteilungen benötigen, müssen wir uns nicht mit messbaren Mengen herumschlagen und verwenden die folgende, einfache Definition eines Wahrscheinlichkeitsraums.

Definition 1.20 Ein *Wahrscheinlichkeitsraum* (Ω, P) besteht aus einer endlichen *Ergebnismenge* $\Omega = \{\omega_1, \dots, \omega_{|\Omega|}\}$ und einer *Wahrscheinlichkeitsverteilung* oder kurz *Verteilung*

$$P\colon \operatorname{Pot}(\Omega) \to [0,1]$$

(wobei $\operatorname{Pot}(\Omega)$ die Potenzmenge von Ω ist), die folgende Eigenschaften erfüllt:

- $P(\emptyset) = 0$,
- $P(\Omega) = 1$,
- $P(T \cup S) = P(T) + P(S)$ für $T, S \subseteq \Omega$ mit $T \cap S = \emptyset$.

Da wir nur den endlichen Fall betrachten, ist P durch die Werte $P(\omega_i) = P(\{\omega_i\})$ (die geschweiften Klammern lassen wir für gewöhnlich weg) für alle $i = 1, \dots, |\Omega|$ bereits eindeutig definiert: für $T \subseteq \Omega$ gilt dann $P(T) = \sum_{\omega_i \in T} P(\omega_i)$. Gilt $P(\omega_i) = \frac{1}{|\Omega|}$ für alle $i = 1, \dots, s$, dann nennt man P eine *Gleichverteilung*.

Die Elemente $\omega_i \in \Omega$ heißen *Ergebnisse*, Teilmengen $T \subseteq \Omega$ werden *Ereignisse* genannt. ◁

Definition 1.21 (Zufallsvariable) Sei (Ω, P) ein Wahrscheinlichkeitsraum, $\mathscr{A}_X = \{a_1, \dots, a_s\}$ eine endliche Menge. Eine Abbildung $X\colon \Omega \to \mathscr{A}$ heißt *Zufallsvariable*. Wir nennen $\mathscr{A}_X$ in unserem Kontext meist das *Alphabet* von X und definieren

$$P_X(A) := P(X \in A) := P(X^{-1}(A))$$

für $A \subseteq \mathscr{A}_X$, wodurch $(\mathscr{A}_X, P_X)$ selbst zu einem Wahrscheinlichkeitsraum wird. Anstelle von $P(X^{-1}(a_i))$ nutzen wir meist eine der bequemeren Notationen

$$P_X(a_i) = P(\{X = a_i\}) = P(X = a_i) = P(a_i) = p_i,$$

schreiben also das X nicht explizit hin, wenn es aus dem Kontext klar ist.

Ist $\mathscr{A}_X \subseteq \mathbb{R}$, nennt man X eine *reelle Zufallsvariable*. Gilt $P(X = x) = \frac{1}{|\mathscr{A}_X|}$ für alle $x \in \mathscr{A}_X$, dann heißt X *gleichverteilt*. ◁

Definition 1.22 (Gemeinsame Verteilung, mehrdimensionale Zufallsvariable) Das Tupel (Ω, P) sei ein Wahrscheinlichkeitsraum und seien $X_i\colon \Omega \to \mathscr{A}_{X_i}$ für $i = 1, \dots, N$ Zufallsvariablen. Diese lassen sich zu einer Zufallsvariable $X\colon \Omega \to \mathscr{A}_X$ zusammenfassen, die durch $X(\omega) = (X_1(\omega), \dots, X_N(\omega))$ definiert ist, also das Alphabet $\mathscr{A}_X = \mathscr{A}_{X_1} \times \cdots \times \mathscr{A}_{X_N}$ und die Verteilung P_X hat, die für $T_1 \subseteq \mathscr{A}_{X_1}, \dots, T_N \subseteq \mathscr{A}_{X_N}$ durch

$$P_X(T_1, \dots, T_N) = P(X^{-1}(T_1, \dots, T_N)) = P(X_1^{-1}(T_1) \cap \cdots \cap X_N^{-1}(T_N))$$

gegeben ist. P_X wird die *gemeinsame Verteilung* von $X_1, \dots, X_N$ genannt. Die Wahrscheinlichkeit eines Ergebnisses $x = (x_1, \dots, x_N) \in \mathscr{A}_X$ lässt sich berechnen durch

$$P_X(X = x) = \sum_{\substack{\omega \in \Omega : \\ X(\omega) = x}} P(\omega).$$

◁

Bemerkung 1.23 Zuweilen werden wir auch den umgekehrten Weg gehen: Ist $X : \Omega \to \mathscr{A}_X$ eine Zufallsvariable mit $\mathscr{A}_X = \mathscr{A}_{X_1} \times \cdots \times \mathscr{A}_{X_N}$, $N \in \mathbb{N}$, ist $\mathscr{A}_X$ also ein N-Tupel (endlicher) Mengen, definieren wir für $i \in \{1, \dots, N\}$ die Zufallsvariable X_i mit Alphabet $\mathscr{A}_{X_i}$ und Verteilung

$$P_{X_i}(T_i) = P_{X_i}(X_i \in T_i) = P(X_i^{-1}(T_i)) = \sum_{\substack{x \in \mathscr{A}_X : \\ x_i \in T_i}} P_X(x), \tag{1.1}$$

die auch als *Marginalverteilung* der X_i bezeichnet wird. ◁

Definition 1.24 (Unabhängigkeit und bedingte Wahrscheinlichkeit) Sei $X : \Omega \to \mathscr{A}_X$ eine Zufallsvariable. Zwei Ereignisse $T, S \subseteq \mathscr{A}_X$ heißen *unabhängig*, wenn gilt

$$P(X \in T \cap S) = P(X \in T)P(X \in S).$$

Die *bedingte Wahrscheinlichkeit von T gegeben S* ist (für $P(S) \neq 0$) definiert als

$$P_X(T \mid S) := P(X \in T \mid X \in S) := \frac{P(X \in T \cap S)}{P(X \in S)}.$$

Zwei Zufallsvariablen $X : \Omega \to \mathscr{A}_X$ und $Y : \Omega \to \mathscr{A}_Y$ mit gemeinsamer Verteilung P_{XY} heißen *unabhängig*, wenn

$$P_{XY}(T, S) = P_X(T) \cdot P_Y(S)$$

für alle $T \subseteq \mathscr{A}_X$ und $S \in \mathscr{A}_Y$ gilt.

Für $S \subseteq \mathscr{A}_Y$ mit $P(S) \neq 0$ definieren wir die *bedingte Verteilung von X gegeben $Y \in S$* durch

$$P_{X|Y \in S}(T) = P(X \in T \mid Y \in S)$$

für $T \subseteq \mathscr{A}_X$. ◁

Beispiel 1.25 Häufig hat man N Zufallsvariablen $X_1, \dots, X_N$, die alle dasselbe Alphabet $\mathscr{A}_{X_i} = \mathscr{A}_X$ und die gleiche Verteilung $P_{X_i} = P_X$ haben und außerdem unabhängig sind, beispielsweise wenn man ein Zufallsexperiment N-mal unter gleichen Voraussetzungen wiederholt und X_i den Ausgang des i-ten Versuchs beschreibt. Wir bezeichnen dann die gemeinsame Zufallsvariable $X_1 \cdots X_N$ mit X^N; ihr Alphabet ist $\mathscr{A}_X^N$ und die Wahrscheinlichkeit eines $x = (x_1, \dots, x_N) \in \mathscr{A}_X^N$ ist $P(x) = \prod_{i=1}^N P_X(x_i)$. ◁

Beispiel 1.26 (3-dimensionale Zufallsvariable) Die Zufallsvariable X bilde ein zufällig aus einem deutschen Wörterbuch ausgewähltes Wortes auf seine ersten drei Buchstaben ab. Dann ist $\mathscr{A}_X = \mathscr{A} \times \mathscr{A} \times \mathscr{A} = \mathscr{A}^3$, wobei $\mathscr{A} = \{\mathtt{a}, \dots, \mathtt{z}, \ddot{\mathtt{a}}, \ddot{\mathtt{o}}, \ddot{\mathtt{u}}, \mathtt{ß}\}$ die Buchstaben des deutschen Alphabets umfasst (wir ignorieren Groß- und Kleinschreibung). Für $i = 1, 2, 3$ entspricht X_i der Abbildung auf den i-ten Buchstaben des ausgewählten Wortes. Die Verteilung P_X hängt vom Inhalt des Wörterbuchs ab; beispielsweise dürfte $P_X(\mathtt{z}, \mathtt{x}, \mathtt{q}) = 0$ sein, $P_X(\mathtt{e}, \mathtt{i}, \mathtt{n})$ dagegen positiv.

Entsprechend Bemerkung 1.23 können wir X in drei Zufallsvariablen X_1, X_2, X_3 aufteilen, wobei X_i dem jeweils i-ten Buchstaben eines Zufallswortes entspricht. Die Verteilung P_{X_i} können wir mit (1.1) bestimmen, z. B. ist

$$P_{X_2}(\mathtt{e}) = P(X_2 = \mathtt{e}) = \sum_{\substack{x \in \mathscr{A}_X: \\ x_2 = \mathtt{e}}} P(X = x) = \sum_{x_1, x_3 \in \mathscr{A}} P(x_1, \mathtt{e}, x_3)$$

gleich dem Anteil der Wörter, deren zweiter Buchstabe ein $\mathtt{e}$ ist. Natürlich sind X_1, X_2, X_3 *nicht* unabhängig, das vorherige Beispiel lässt sich hier also nicht anwenden. ◁

Aufgabe 1.27 (Satz von Bayes) Seien (Ω, P) ein Wahrscheinlichkeitsraum und $X : \Omega \to \mathscr{A}_X$ und $Y : \Omega \to \mathscr{A}_Y$ zwei Zufallsvariablen. Zeigen Sie:

1. *Satz von Bayes*:
$$P_{X|Y \in S}(T) = P_{Y|X \in T}(S) \frac{P_X(T)}{P_Y(S)},$$
für alle Ereignisse $T \subseteq \mathscr{A}_X$, $S \subseteq \mathscr{A}_Y$ mit $P_X(T), P_Y(S) > 0$.
2. X und Y sind genau dann unabhängig, wenn für alle $T \in \mathscr{A}_X$ und $S \in \mathscr{A}_Y$ gilt:
$$P_Y(S) = 0 \text{ oder } P_{X|Y \in S}(T) = P_X(T).$$ ◁

1.2.2. Satz von Shannon

Enthält der Klartextraum $\mathscr{P}$ Wörter einer menschenverständlichen Sprache, so treten verschiedene Klartexte mit unterschiedlicher Wahrscheinlichkeit auf. Die Klartexte bzw. Klartextteile „Zeit“, „Treffpunkt“, „Vertrag“ werden in der Regel deutlich häufiger genutzt als „Alliteration“, „Onomatopoesie“, „Pleonasmus“. Somit ist auch die Wahrscheinlichkeit für eine Verschlüsselung von „Zeit“, „Treffpunkt“ oder „Vertrag“ deutlich größer als die Wahrscheinlichkeit für eine Verschlüsselung von „Alliteration“, „Onomatopoesie“ oder „Pleonasmus“.

Wenn ein Kryptosystem in der Praxis eingesetzt wird, folgen nicht nur die Klartext einer Wahrscheinlichkeitsverteilung, sondern auch die Schlüssel: Ein genutzter Verschlüsselungsschlüssel $e \in \mathscr{K}$ fällt ja nicht vom Himmel, er muss gewählt werden. Unterschiedliche Schlüssel können unterschiedliche (Nutzungs-)Wahrscheinlichkeiten haben, der Schlüssel „sdklfjlg“ kann beispielsweise mit einer größeren Wahrscheinlichkeit gewählt werden als „eigbswöll“.

Klartext- und Schlüsselraum gemeinsam betrachtet, gibt es also eine Wahrscheinlichkeit dafür, dass ein Klartext $p \in \mathcal{P}$ mit dem Schlüssel $e \in \mathcal{K}$ verschlüsselt wird. Diesen Gedanken formalisieren wir nun.

Im Rest dieses Abschnitts sei $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ ein Secret-Key-Kryptosystem, wobei wir annehmen, dass $\mathcal{P}$, $\mathcal{C}$ und $\mathcal{K}$ endliche Mengen sind.

Wir definieren $\Omega := \mathcal{P} \times \mathcal{K}$ und

$$P : \mathrm{Pot}(\Omega) \to [0,1],\ (p,e) \mapsto P(p,e),$$

als die Verteilung, die die Wahrscheinlichkeit angibt, dass p durch e verschlüsselt wird. (Ω, P) wird so zu einem Wahrscheinlichkeitsraum. Wir betrachten die Zufallsvariablen $X_{\mathcal{P}} : \Omega \to \mathcal{P}$, $(p,e) \mapsto p$ und $X_{\mathcal{K}} : \Omega \to \mathcal{K}$, $(p,e) \mapsto e$, die durch die beiden Projektionen gegeben sind. Die zugehörigen Verteilungen $P_{X_{\mathcal{P}}} : \mathrm{Pot}(\mathcal{P}) \to [0,1]$ und $P_{X_{\mathcal{K}}} : \mathrm{Pot}(\mathcal{K}) \to [0,1]$ geben an, mit welcher Wahrscheinlichkeit ein Klartext p bzw. ein Verschlüsselungsschlüssel e auftritt:

$$P_{X_{\mathcal{P}}}(p) = P(X_{\mathcal{P}} = p) = P(\{(p,e') \in \Omega \mid e' \in \mathcal{K}\}) = \sum_{e' \in \mathcal{K}} P(p,e')$$

und

$$P_{X_{\mathcal{K}}}(e) = P(X_{\mathcal{K}} = e) = P(\{(p',e) \in \Omega \mid p' \in \mathcal{P}\}) = \sum_{p' \in \mathcal{P}} P(p',e).$$

Wir treffen folgende (vertretbare) Annahmen:

1. $P_{X_{\mathcal{P}}}(p) > 0$ für alle $p \in \mathcal{P}$, d. h. die Wahrscheinlichkeit, dass der Klartext p verschlüsselt wird, ist größer null.
2. $P_{X_{\mathcal{K}}}(e) > 0$ für alle $e \in \mathcal{K}$, d. h. die Wahrscheinlichkeit, dass der Schlüssel e benutzt wird, ist größer null.
3. $\mathcal{P} \times \mathcal{K} \to \mathcal{C}$, $(p,e) \mapsto E_e(p)$ ist surjektiv, d. h. jeder Chiffretext hat ein Urbild im Klartextraum.
4. Die Zufallsvariablen $X_{\mathcal{P}} : \Omega \to \mathcal{P}$, $(p,e) \mapsto p$ und $X_{\mathcal{K}} : \Omega \to \mathcal{K}$, $(p,e) \mapsto e$ sind unabhängig, d. h. die zusammengefasste Zufallsvariable $X_{\mathcal{P}}X_{\mathcal{K}} : \Omega \to \mathcal{P} \times \mathcal{K}$ hat die gemeinsame Verteilung

   $$P_{X_{\mathcal{P}}X_{\mathcal{K}}}(p,e) = P(\{X_{\mathcal{P}} = p\} \cap \{X_{\mathcal{K}} = e\}) = P_{X_{\mathcal{P}}}(p) \cdot P_{X_{\mathcal{K}}}(e).$$

   Anschaulich gesprochen ist die Wahl des Klartextes p also unabhängig von der Wahl des Schlüssels e.

Bemerkung 1.28 Bilden wir zuerst die beiden Projektionen $X_{\mathcal{P}} : \Omega \to \mathcal{P}$ und $X_{\mathcal{K}} : \Omega \to \mathcal{K}$ und dann die zusammengefasste Zufallsvariable $X_{\mathcal{P}}X_{\mathcal{K}} : \Omega \to \mathcal{P} \times \mathcal{K} = \Omega$, so erhalten wir die identische Abbildung $X_{\mathcal{P}}X_{\mathcal{K}} : \Omega \to \Omega$, $(p,e) \mapsto (p,e)$ als Zufallsvariable. Daher gilt

$$P_{X_{\mathcal{P}}X_{\mathcal{K}}}(p,e) = P(X_{\mathcal{P}}X_{\mathcal{K}} = (p,e)) = P(p,e),$$

somit $P = P_{X_{\mathcal{P}}X_{\mathcal{K}}}$ und laut obiger Annahme $P(p,e) = P_{X_{\mathcal{P}}}(p) \cdot P_{X_{\mathcal{K}}}(e)$. ◁

Bemerkung 1.29 Die Verteilung der Zufallsvariablen $X_{\mathscr{C}} : \Omega \to \mathscr{C}$, $(p, e) \mapsto E_e(p)$ ist gegeben durch

$$P_{X_{\mathscr{C}}}(c) = P(X_{\mathscr{C}} = c) = \sum_{\substack{(p,e)\in\Omega:\\ E_e(p)=c}} P(p, e)$$

für alle $c \in \mathscr{C}$. ◁

Definition 1.30 (Perfekte Sicherheit, Shannon 1949) Ein Kryptosystem heißt *perfekt sicher für P* (oder einfach *perfekt für P*), wenn $X_{\mathscr{P}}$ und $X_{\mathscr{C}}$ unabhängig sind, d. h. wenn für alle $p \in \mathscr{P}$ und alle $c \in \mathscr{C}$ gilt

$$P_{X_{\mathscr{C}}}(c) = 0 \text{ oder } P_{X_{\mathscr{P}}|X_{\mathscr{C}}=c}(p) = P_{X_{\mathscr{P}}}(p),$$

oder äquivalent

$$P_{X_{\mathscr{P}}}(p) = 0 \text{ oder } P_{X_{\mathscr{C}}|X_{\mathscr{P}}=p}(c) = P_{X_{\mathscr{C}}}(c).$$

Intuitiv bedeutet diese Eigenschaft, dass die Wahrscheinlichkeit eines bestimmten Klartextes unabhängig vom Chiffretext ist. Der Chiffretext gibt also keine Information über den zugehörigen Klartext preis. ◁

Aufgabe 1.31 Zeigen Sie, dass die beiden Bedingungen in der Definition der perfekten Sicherheit äquivalent sind. ◁

Um den Satz von Shannon 1.35 zu beweisen, brauchen wir folgende Notation und folgendes Lemma.

Notation 1.32 ($\mathscr{S}_{p,c}$) Seien $p \in \mathscr{P}$, $c \in \mathscr{C}$. Mit $\mathscr{S}_{p,c} = \{e \in \mathscr{K} \mid E_e(p) = c\}$ bezeichnen wir die Menge, die genau die Schlüssel enthält, die p zu c verschlüsseln. ◁

Lemma 1.33 *Seien $p \neq p'$ zwei Klartexte und c ein Chiffretext in einem perfekten Kryptosystem. Dann gilt:*

1. *$\mathscr{S}_{p,c} \neq \emptyset$, d. h. es existiert ein Schlüssel e, der p zu c verschlüsselt.*
2. *$\mathscr{S}_{p,c} \cap \mathscr{S}_{p',c} = \emptyset$, d. h. wenn ein Schlüssel e den Klartext p zu c verschlüsselt, dann kann er nicht auch p' zu c verschlüsseln.* ◁

Beweis. Zu 1.: Es gilt

$$P_{X_{\mathscr{C}}}(c) = \sum_{\substack{(p,e)\in\Omega:\\ E_e(p)=c}} P(p, e) = \sum_{\substack{(p,e)\in\Omega:\\ E_e(p)=c}} P_{X_{\mathscr{P}}}(p) \cdot P_{X_{\mathscr{K}}}(e) > 0,$$

da wir in diesem Abschnitt angenommen haben, dass jeder Chiffretext ein Urbild im Klartextraum hat und dass jeder Schlüssel und jeder Klartext benutzt wird. Aus der perfekten Sicherheit des Kryptosystems folgt für alle $p \in \mathscr{P}$ die Eigenschaft

$$P_{X_{\mathscr{C}}|X_{\mathscr{P}}=p}(c) = P_{X_{\mathscr{C}}}(c) > 0,$$

d. h. die Wahrscheinlichkeit, dass der Klartext p zu c verschlüsselt wird, ist positiv. Insbesondere existiert somit ein $e \in \mathscr{K}$, das $E_e(p) = c$ und somit $e \in \mathscr{S}_{p,c}$ erfüllt.

Zu 2.: Angenommen $e \in \mathscr{S}_{p,c} \cap \mathscr{S}_{p',c}$. Dann gilt $E_e(p) = c = E_e(p')$ und wir folgern aus der Injektivität von E_e (siehe Aufgabe 1.8) $p = p'$; ein Widerspruch zur Voraussetzung $p \neq p'$. □

Da die Verschlüsselungsfunktionen $E_e : \mathscr{P} \to \mathscr{C}$ injektiv sind, siehe Aufgabe 1.8, ist die Anzahl der Chiffretexte mindestens so groß wie die Anzahl der Klartexte. In perfekt sicheren Kryptosystemen gilt zusätzlich, dass auch die Anzahl der Schlüssel mindestens so groß wie die Anzahl der Klartexte ist.

Aufgabe 1.34 Zeigen Sie: In einem perfekten Kryptosystem gilt $|\mathscr{P}| \leq |\mathscr{K}|$. ◁

Unter der zusätzlichen Voraussetzung $|\mathscr{K}| = |\mathscr{P}| = |\mathscr{C}|$ gelang *Shannon* im Jahr 1949 eine vollständige Charakterisierung perfekt sicherer Kryptosysteme.

Satz 1.35 (Shannon) *Ein Kryptosystem erfülle*

$$|\mathscr{K}| = |\mathscr{P}| = |\mathscr{C}|.$$

Das System ist genau dann perfekt sicher, wenn die folgenden beiden Bedingungen erfüllt sind:

1. *$X_{\mathscr{K}}$ ist gleichverteilt, d. h. jeder Schlüssel tritt mit der gleichen Wahrscheinlichkeit auf.*
2. *Für jeden Klartext $p \in \mathscr{P}$ und für jeden Chiffretext $c \in \mathscr{C}$ gibt es genau einen Schlüssel $e \in \mathscr{K}$, der p zu $E_e(p) = c$ verschlüsselt.* ◁

Beweis. Wir nehmen an, dass das Kryptosystem perfekt sicher ist, und zeigen zuerst, dass dann auch 2. erfüllt ist:

Sei $c \in \mathscr{C}$ ein Chiffretext. Dann gilt mit Lemma 1.33

$$\begin{aligned}
|\mathscr{K}| &\geq \left|\left(\bigcup_{p\in\mathscr{P}} \mathscr{S}_{p,c}\right)\right| && \text{alle } \mathscr{S}_{p,c} \text{ sind in } \mathscr{K} \text{ enthalten} \\
&= \sum_{p\in\mathscr{P}} |\mathscr{S}_{p,c}| && \text{die Mengen } \mathscr{S}_{p,c} \text{ sind paarweise disjunkt} \\
&\geq |\mathscr{P}| && |\mathscr{S}_{p,c}| \geq 1 \\
&= |\mathscr{K}| && \text{nach Voraussetzung}
\end{aligned} \tag{1.2}$$

Alle Ungleichungen sind demnach Gleichungen und wir erhalten

$$\sum_{p\in\mathscr{P}} |\mathscr{S}_{p,c}| = |\mathscr{P}|.$$

Da alle $\mathscr{S}_{p,c}$ nichtleer sind (Lemma 1.33), folgt $|\mathscr{S}_{p,c}| = 1$ für alle $p \in \mathscr{P}$. Es gibt also genau einen Schlüssel, der p zu c verschlüsselt, und 2. ist erfüllt.

Nun zeigen wir, dass für perfekte Kryptosysteme auch 1. gilt:

Sei wieder $c \in \mathscr{C}$ ein Chiffretext. Für $p \in \mathscr{P}$ bezeichnen wir mit $e(p,c)$ das eindeutige Element von $\mathscr{S}_{p,c}$, d. h. den eindeutigen Schlüssel, der p zu c verschlüsselt. Mit Aufgabe 1.36 folgt

$$\mathscr{K} = \left|\dot{\bigcup}_{p\in\mathscr{P}} \mathscr{S}_{p,c}\right| = \{e(p,c) \mid p \in \mathscr{P}\}.$$

Da $X_{\mathscr{P}}$ und $X_{\mathscr{K}}$ unabhängig sind, gilt für alle $p \in \mathscr{P}$

$$\begin{aligned} P_{X_{\mathscr{C}}|X_{\mathscr{P}}=p}(c) &= \frac{P(\{X_{\mathscr{C}} = c\} \cap \{X_{\mathscr{P}} = p\})}{P(X_{\mathscr{P}} = p)} & &= \frac{P(\{(p,e)|e \in \mathscr{K}, E_e(p) = c\})}{P(X_{\mathscr{P}} = p)} \\ &= \frac{P(p, e(p,c))}{P(X_{\mathscr{P}} = p)} & &= \frac{P(\{X_{\mathscr{P}} = p\} \cap \{X_{\mathscr{K}} = e(p,c)\})}{P(X_{\mathscr{P}} = p)} \\ &= \frac{P(\{X_{\mathscr{P}} = p\}) \cdot P(\{X_{\mathscr{K}} = e(p,c)\})}{P(X_{\mathscr{P}} = p)} k & &= P_{X_{\mathscr{K}}}(e(p,c)). \end{aligned}$$

(1.3)
Da das Kryptosystem perfekt ist, folgt

$$P_{X_{\mathscr{C}}}(c) = P_{X_{\mathscr{C}}|X_{\mathscr{P}}=p}(c) = P_{X_{\mathscr{K}}}(e(p,c))$$

für alle $p \in \mathscr{P}$. Da für jeden Schlüssel $e \in \mathscr{K}$ ein $p \in \mathscr{P}$ existiert, das $e = e(p,c)$ erfüllt (siehe Aufgabe 1.36), gilt auch

$$P_{X_{\mathscr{C}}}(c) = P_{X_{\mathscr{K}}}(e)$$

für alle $e \in \mathscr{K}$. Da $c \in \mathscr{C}$ fest gewählt war, ist insbesondere $P_{X_{\mathscr{K}}}(e)$ für alle $e \in \mathscr{K}$ gleich, d. h. alle Schlüssel $e \in \mathscr{K}$ sind gleich wahrscheinlich und $X_{\mathscr{K}}$ ist gleichverteilt.

Nun zeigen wir die Rückrichtung und nehmen an, dass 1. und 2. erfüllt sind:

Für $p \in \mathscr{P}$ und $c \in \mathscr{C}$ bezeichnen wir (wie oben) mit $e(p,c)$ den eindeutigen Schlüssel, der p zu c verschlüsselt. Mit dem Satz von Bayes und Gleichung 1.3 folgt für alle $p \in \mathscr{P}$, $c \in \mathscr{C}$

$$P_{X_{\mathscr{P}}|X_{\mathscr{C}}=c}(p) = P_{X_{\mathscr{P}}}(p)\frac{P_{X_{\mathscr{C}}|X_{\mathscr{P}}=p}(c)}{P_{X_{\mathscr{C}}}(c)} = P_{X_{\mathscr{P}}}(p)\frac{P_{X_{\mathscr{K}}}(e(p,c))}{\sum_{q\in\mathscr{P}} P_{X_{\mathscr{P}}}(q)P_{X_{\mathscr{K}}}(e(q,c))}.$$

Nach Voraussetzung sind alle Schlüssel gleich wahrscheinlich, d. h. $P_{X_{\mathscr{K}}}(e(q,c)) = \frac{1}{|\mathscr{K}|}$ für alle $q \in \mathscr{P}$, außerdem gilt $\sum_{q\in\mathscr{P}} P_{X_{\mathscr{P}}}(q) = 1$. Daraus folgt

$$\frac{P_{X_{\mathscr{K}}}(e(p,c))}{\sum_{q\in\mathscr{P}} P_{X_{\mathscr{P}}}(q)P_{X_{\mathscr{K}}}(e(q,c))} = \frac{\frac{1}{|\mathscr{K}|}}{\frac{1}{|\mathscr{K}|}\sum_{q\in\mathscr{P}} P_{X_{\mathscr{P}}}(q)} = 1.$$

Insgesamt erhalten wir

$$P_{X_{\mathscr{P}}|X_{\mathscr{C}}=c}(p) = P_{X_{\mathscr{P}}}(p)$$

für alle $p \in \mathscr{P}$ und alle $c \in \mathscr{C}$; das Kryptosystem ist also perfekt sicher. □

Aufgabe 1.36 Sei $|\mathscr{P}| = |\mathscr{K}| = |\mathscr{C}|$ und sei $c \in \mathscr{C}$ ein Chiffretext. Zeigen Sie, dass für jeden Schlüssel $e \in \mathscr{K}$ ein Klartext $p \in \mathscr{P}$ existiert, der mit dem Schlüssel e zu c verschlüsselt wird, d. h. der $E_e(p) = c$ erfüllt. ◁

Beispiel 1.37 (Vernam-One-Time-Pad) Das bekannteste Kryptosystem, dessen perfekte Sicherheit man mit dem Satz von Shannon beweisen kann, ist das *Vernam-One-Time-Pad*. Es wurde 1917 von Gilbert Vernam erfunden und patentiert. Es ist wie folgt aufgebaut:

Sei $n \in N$ und $\mathcal{P} = \mathcal{K} = \mathcal{C} = \mathbb{Z}_2^n$. (Zur Erinnerung: Es gilt $-p = p$ für alle $p \in \mathbb{Z}_2^n$.) Die Verschlüsselungsfunktion E_e zum Schlüssel $e \in \mathbb{Z}_2^n$ ist gegeben durch die komponentenweise Addition

$$E_e : \mathbb{Z}_2^n \to \mathbb{Z}_2^n,\ p \mapsto p + e.$$

Die zum Schlüssel $d \in \mathbb{Z}_2$ gehörige Entschlüsselungsfunktion $D_d : \mathbb{Z}_2^n \to \mathbb{Z}_2^n, c \mapsto c + e$, sieht genauso aus. Damit gilt für alle $k \in \mathcal{K}$

$$D_k \circ E_k = \mathrm{id}_{\mathcal{P}} : \mathbb{Z}_2^n \to \mathbb{Z}_2^n,\ p \mapsto D_k(E_k(p)) = p + k + k = p.$$

Um einen Klartext $p \in \mathbb{Z}_2^n$ zu verschlüsseln, wird gemäß einer Gleichverteilung ein Schlüssel e aus $\mathbb{Z}_2^n$ gewählt und der Chiffretext $c := E_e(p) = p + e$ berechnet. Da $(Z_2^n, +)$ eine Gruppe ist, gibt es genau einen Schlüssel $e = e(p, c) \in \mathbb{Z}_2^n$, der $p + e = c$ erfüllt, nämlich $e = c + (-p) = c + p$. Mit dem Satz von Shannon folgt, dass das Vernam-One-Time-Pad perfekt sicher ist.

Auch wenn das Vernam-One-Time-Pad perfekt sicher ist, ist es nicht effizient. Jeder Schlüssel kann nur einmal verwendet werden und für jeden Klartextblock der Länge n muss ein neuer Schlüssel ausgetauscht werden. Daher kommt der Name One-Time-Pad. Wird ein Schlüssel mehr als einmal benutzt, sind die benutzten Schlüssel nicht mehr gleichverteilt und das One-Time-Pad ist nicht mehr perfekt sicher.

Das Vernam-One-Time-Pad hat eine weitere Schwäche: Wenn ein Angreifer weiß, an welcher Stelle des Chiffretexts eine Zahl steht, kann er diese Zahl beliebig verändern, ohne dass der Sinn der Nachricht zerstört wird. Bei der Übermittlung von Überweisungen könnte beispielsweise der Überweisungsbetrag geändert werden. ◁

Bemerkung 1.38 Der im zweiten Teil der Vorlesung über Codierungstheorie eingeführte Begriff der *Entropie* erlaubt eine weitere Charakterisierung von perfekt sicheren Kryptosystemen, bei der nicht angenommen werden muss, dass $|\mathcal{P}| = |\mathcal{C}|$ gilt. Auch diese Charakterisierung wurde von Shannon bewiesen. ◁

2. Secret-Key-Kryptosysteme

Im einführenden Vortrag wurden historische Secret-Key-Kryptosysteme beschrieben. Alle diese Systeme haben sich im Laufe der Zeit als unsicher herausgestellt und können heutzutage gebrochen werden können. Daraufhin wurde im vorhergehenden Kapitel der Frage nachgegangen, ob es überhaupt sichere Kryptosysteme gibt. Der Begriff der perfekten Sicherheit wurde eingeführt, und am Beispiel des Vernam-One-Time-Pads wurde gezeigt, dass Kryptosysteme existieren, die beweisbar perfekt sind. In der Praxis haben sich perfekte Kryptosysteme allerdings als ineffizient und unpraktikabel herausgestellt. In diesem Kapitel werden wir das Secret-Key-Kryptosystem AES vorstellen, das heutzutage genutzt wird und als sicher gilt.

Vom US-amerikanischen *National Institutue of Standards and Technology* (NIST) wurde 1977 der Data Encryption Standard (DES) als offizieller Verschlüsselungsstandard eingeführt. Seitdem wurde DES weltweit eingesetzt. Aufgrund schnellerer Computer galt aber auch DES seit Ende der 90er nicht mehr als sicher. Daraufhin wurde 1997 vom NIST ein Auswahlprozess für einen Nachfolger begonnen. Den Wettbewerb gewann die Rijndael-Chiffre, die nach ihren beiden Erfindern Joan Daemen und Vincent Rijmen benannt ist. Seit 2001 ist AES standardisiert, wird weltweit eingesetzt und gilt als sicher. AES wird u.a. genutzt bei der Datenübertragung mit Wireless LAN, dem Netzwerkprotokoll SSH und Skype sowie bei verschlüsselten `rar`-Archiven.

Sowohl DES als auch AES haben die Struktur einer Blockchiffre. Im Folgenden werden Blockchiffren eingeführt und die Funktionsweise von AES erläutert.

2.1. Blockchiffren

Definition 2.1 (Permutation) Sei M eine Menge. Eine *Permutation von M* ist eine bijektive Abbildung $f : M \to M$. Die Menge aller Permutationen von M wird mit $\mathbb{S}(M)$ bezeichnet, die Menge aller Permutationen auf $\{1, \dots, n\}$ mit $\mathbb{S}_n$. Zur Erinnerung: $(\mathbb{S}(M), \circ)$ ist eine Gruppe, wobei $\circ$ für die Komposition von Abbildungen steht. ◁

Beispiel 2.2 Sei $n \in \mathbb{N}$. Die Abbildung $\sigma : \mathbb{Z}_n \to \mathbb{Z}_n$, $a \mapsto -a$, ist bijektiv und damit eine Permutation $\sigma \in \mathbb{S}(\mathbb{Z}_n)$. ◁

Aufgabe 2.3 Sei M eine endliche Menge und $f : M \to M$ injektiv. Zeigen Sie, dass f dann auch surjektiv (und folglich eine Permutation) ist. ◁

Definition 2.4 (Blockchiffre) Seien $n \in \mathbb{N}$ und A ein Alphabet. Ein Kryptosystem heißt *Blockchiffre der Länge $n \in \mathbb{N}$ auf A*, wenn $\mathcal{P} = \mathcal{C} = A^n$ gilt, d. h. der Klartext- und der Chiffretextraum sind durch die Menge aller Wörter der Länge n auf dem Alphabet A gegeben. ◁

Satz 2.5 *Die Verschlüsselungsfunktionen einer Blockchiffre sind Permutationen von A^n.* ◁

Beweis. Gemäß Aufgabe 1.8 sind Verschlüsselungsfunktionen $\mathcal{P} = A^n \to \mathcal{C} = A^n$ injektiv, außerdem ist der Klartext- und Chiffretextraum A^n endlich. Mit Aufgabe 2.3 folgt, dass die Verschlüsselungsfunktionen Permutationen sind. □

Beispiel 2.6 Ein Beispiel für eine Blockchiffre ist die Permutationschiffre auf dem Alphabet A mit Blocklänge $n \in \mathbb{N}$. Der Schlüsselraum ist durch $\mathcal{K} = \mathbb{S}_n$ gegeben mit $|\mathbb{S}_n| = n!$. Die zu $\pi \in \mathbb{S}_n$ gehörige Verschlüsselungsfunktion ist

$$E_\pi : A^n \to A^n, \quad (v_1, \dots, v_n) \mapsto (v_{\pi(1)}, \dots, v_{\pi(n)});$$

sie ist gleich der zu π gehörigen Entschlüsselungsfunktion $D_\pi : A^n \to A^n$, d. h. es gilt $D_\pi = E_\pi$.

Konkreter: Das Alphabet sei $A = \{0, \dots, 9\}$, die Schlüssellänge $n = 4$ und wir betrachten die Permutation

$$\pi = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{smallmatrix}\right).$$

Die zu π inverse Permutation ist

$$\pi^{-1} = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{smallmatrix}\right).$$

Das Wort $v = (3, 6, 2, 7) \in A^4$ (man beachte, dass die Länge des Wortes gleich der „Länge“ der Permutation ist) wird mit dem Schlüssel π zu

$$w := E_\pi(v) = E_\pi(3, 6, 2, 7) = (v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)}, v_{\pi(4)}) = (v_3, v_2, v_4, v_1) = (2, 6, 7, 3)$$

verschlüsselt und mit $D_{\pi^{-1}}$ wieder zu

$$D_{\pi^{-1}}(E_\pi(v)) = D_{\pi^{-1}}(w) = D_{\pi^{-1}}(2, 6, 7, 3) = (w_4, w_2, w_1, w_3) = (3, 6, 2, 7)$$

entschlüsselt.

Bei Permutationschiffren ist der i-te Buchstabe des Chiffretexts durch den $\pi(i)$-ten Buchstaben des Klartextes gegeben, wie man im Beispiel gut sieht. Durch unterschiedliche Wahlen von $\pi \in \mathbb{S}_n$ kann ein Buchstabe des Klartextes somit jeden Buchstaben des Chiffretexts beeinflussen. ◁

Bemerkung 2.7 (Beginn einer Kryptoanalyse von Blockchiffren) Eine Methode Blockchiffren auf ihre Unsicherheit zu untersuchen, besteht darin, die den Verschlüsselungsfunktionen zugrundeliegenden Permutationen zu studieren. Ist beispielsweise die Ordnung $k \in \mathbb{N}$ der Permutation $\sigma \in \mathbb{S}(\mathbb{Z}_m^n)$ klein (d. h. $\sigma^k = \mathrm{id}$ für ein recht kleines k und damit $\sigma^k(p) = p$ für alle $p \in \mathcal{P}$), kann man sie durch iterierte Anwendung entschlüsseln: Es gilt $\sigma^k(p) = p$ für alle Klartexte $p \in \mathbb{Z}_m^n$. Ein Angriff würde z. B. darin bestehen, eine Liste aller Permutationen mit „kleiner“ Ordnung zu erstellen und diese nacheinander wiederholt auf den Chiffretext anzuwenden. Um eine sichere Blockchiffre aufzusetzen, sollte man daher nur Permutationen mit einer hinreichend großen Ordnung verwenden. ◁

Affin lineare Blockchiffren

Auch wenn die heute verbreiteten Blockchiffren als sicher gelten, gibt es Blockchiffren, die relativ leicht gebrochen werden können. Eine Klasse solcher einfach zu brechenden Blockchiffren sind affin lineare Blockchiffren. Wir führen diese Chiffren ein und stellen eine erfolgversprechende Angriffsstrategie vor.

Zuerst wiederholen wir einige Begriffe der Linearen Algebra. Sie sollten Ihnen aus der Vorlesung zu Linearer Algebra im Bachelor bereits bekannt sein.

Sei R ein Ring und $k, n \in \mathbb{N}$. Dann ist eine *$k \times n$-Matrix über R* ein rechteckiges Schema

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,n} \end{pmatrix}.$$

mit k Zeilen, n Spalten und Einträgen $a_{i,j} \in R$. Wir schreiben auch $A = (a_{i,j})$. Die Menge aller $k \times n$-Matrizen über k wird mit $R^{k \times n}$ bezeichnet. Man kann Matrizen komponentenweise addieren und sie multiplizieren, wenn ihre Formate kompatibel sind. Bezeichnen wir mit $A \cdot B = (c_{i,l}) \in R^{k \times n}$ das Produkt von $A = (a_{i,j}) \in R^{k \times s}$ und $B = (b_{j,l}) \in R^{s \times n}$, so gilt

$$c_{i,l} = \sum_{j=1}^{k} a_{i,j} \cdot b_{j,l}.$$

Mit diesen beiden Operationen werden die quadratischen Matrizen $(R^{n \times n}, +, \cdot)$ zu einem (in der Regel nicht-kommutativen) Ring mit der Nullmatrix als neutralem Element der Addition und der Einheitsmatrix (Diagonalmatrix mit Einsen auf der Diagonalen) als neutralem Element der Multiplikation.

Die *Determinante* ist eine Abbildung $\det : R^{n \times n} \to R$. Sie hat die Eigenschaft, dass eine Matrix $A \in R^{n \times n}$ genau dann *invertierbar* in $R^{n \times n}$ ist, wenn ihre Determinante $\det(A) \in R$ eine Einheit in R ist, d. h. $\det(A)$ ist invertierbar in R. Für das Rechnen mit und die Berechnung von Determinanten von Matrizen über einem Ring R gelten die gleichen Regeln wie für Matrizen über $\mathbb{R}$ und $\mathbb{Q}$ (Sarrus für 3×3-Matrizen, Entwicklungsregel von Laplace, Multilinearität, $\det(A \cdot B) = \det(A) \cdot \det(B), \ldots$).

Eine Matrix $A \in R^{k \times n}$ definiert eine Abbildung

$$f_A : R^n \to R^k, \quad x \mapsto A \cdot x.$$

Diese Abbildung f_A ist *linear*, d. h. sie erfüllt $f_A(x + y) = f_A(x) + f_A(y)$ für alle $x, y \in R^n$. Umgekehrt existiert für jede lineare Abbildung $f : R^n \to R^k$ genau eine Matrix $A \in R^{k \times n}$, die $f = f_A$ erfüllt.

Eine Abbildung $f : R^n \to R^k$ heißt *affin linear*, wenn eine Matrix $A \in R^{k \times n}$ und ein Vektor $b \in R^k$ existieren, so dass für alle $x \in R^n$ gilt

$$f(x) = A \cdot x + b.$$

Definition 2.8 ((Affin) lineare Blockchiffre) Eine Blockchiffre mit Blocklänge $n \in \mathbb{N}$ heißt *affin linear*, wenn ihr Alphabet durch $\mathbb{Z}_m$ für ein $m \in \mathbb{N}$ gegeben ist und wenn alle Verschlüsselungsfunktionen $\mathbb{Z}_m^n \to \mathbb{Z}_m^n$ affin linear sind. Sind diese sogar linear, heißt die Blockchiffre *linear*. ◁

Jedes Paar (A, b) aus einer Matrix $A \in \mathbb{Z}_m^{n\times n}$, $m, n \in \mathbb{N}$, und einem Vektor $b \in \mathbb{Z}_m^n$ definiert eine affin lineare Abbildung $\mathbb{Z}_m^n \to \mathbb{Z}_m^n, p \mapsto Ap + b$. Wenn A invertierbar ist, kommt diese affin lineare Abbildung als Verschlüsselungsfunktion einer Blockchiffre mit Alphabet $\mathbb{Z}_m$ und Blocklänge n in Frage. (Ist A nicht invertierbar, dann kann Eigenschaft 5 in Definition 1.5 eines Kryptosystems nicht erfüllt werden.)

Seien beispielsweise

$$A = \begin{pmatrix} [1] & [0] & [1] \\ [0] & [1] & [0] \\ [1] & [0] & [1] \end{pmatrix} \in \mathbb{Z}_2^{3\times 3} \quad \text{und} \quad b = \begin{pmatrix} [0] \\ [1] \\ [0] \end{pmatrix} \in \mathbb{Z}_2^3.$$

Dann würde $x = ([1], [1], [0])^\top \in \mathbb{Z}_2^3 = \mathcal{P}$ zu

$$Ax + b = \begin{pmatrix} [1] & [0] & [1] \\ [1] & [0] & [1] \\ [1] & [0] & [1] \end{pmatrix} \cdot \begin{pmatrix} [1] \\ [1] \\ [0] \end{pmatrix} + \begin{pmatrix} [0] \\ [1] \\ [0] \end{pmatrix} = \begin{pmatrix} [1] \\ [0] \\ [1] \end{pmatrix} \in \mathbb{Z}_2^3 = \mathcal{C}$$

verschlüsselt.

Man kann auch den umgekehrten Weg gehen: Sei $\pi \in \mathbb{S}(\mathbb{Z}_m^n)$ eine Verschlüsselungsfunktion einer affin linearen Blockchiffre. Dann ist π nicht nur eine Permutation auf $\mathbb{Z}_m^n$, sondern auch eine affin lineare Abbildung. Dementsprechend existieren eine invertierbare Matrix $A \in \mathbb{Z}_m^{n\times n}$ und ein Vektor $b \in \mathbb{Z}_m^n$, die $\pi(p) = Ap + b$ für alle $p \in \mathbb{Z}_m^n$ erfüllen.

Affin lineare Verschlüsselungsfunktionen auf $\mathbb{Z}_m^n$ stehen also in einer 1-zu-1-Beziehung zu Paaren (A, b) aus einer invertierbaren Matrix $A \in \mathbb{Z}_m^{n\times n}$ und einem Vektor $b \in \mathbb{Z}_m^n$.

Aufgabe 2.9 Konzipieren Sie eine Verschlüsselungsfunktion einer affin linearen Blockchiffre der Blocklänge 4 auf $\mathbb{Z}_3$, die folgende Eigenschaft erfüllt:

1. Der erste Buchstabe des Klartexts $p = (p_1, \dots, p_4) \in \mathbb{Z}_3^4$ beeinflusst genau den dritten Buchstaben des Chiffretexts. Anders ausgedrückt: Ändert sich nur (!) der erste Buchstabe des Klartexts, ändert sich nur (!) der dritte Buchstabe des Chiffretexts.
2. Der erste Buchstabe des Klartexts $p = (p_1, \dots, p_4) \in \mathbb{Z}_3^4$ beeinflusst jeden Buchstaben des Chiffretexts. Anders ausgedrückt: Ändert sich nur (!) der erste Buchstabe des Klartexts, ändert sich jeder Buchstabe des Chiffretexts.
3. Jeder Buchstabe des Klartexts $p = (p_1, \dots, p_4) \in \mathbb{Z}_3^4$ beeinflusst jeden Buchstaben des Chiffretexts. Anders ausgedrückt: Ändert sich genau ein (!) Buchstabe des Klartexts, während alle restlichen festgehalten werden, ändert sich jeder Buchstabe des Chiffretexts. ◁

Bemerkung 2.10 (Entschlüsselungsfunktion einer affin linearen Blockchiffre) Sei $E_e : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, x \mapsto E_e(x) = Ax + b$ eine Verschlüsselungsfunktion einer affin linearen Blockchiffre. Wir bestimmen die zugehörige Entschlüsselungsfunktion:

$E_e \in \mathbb{S}(Z_m^n)$ ist eine Permutation und somit invertierbar, also ist $E_e^{-1} \in \mathbb{S}(\mathbb{Z}_m^n)$ die zugehörige Entschlüsselungsfunktion. Für alle $c \in \mathbb{Z}_m^n$ ist sie gegeben durch

$$E_e^{-1}(c) = A^{-1}(c - b) = A^{-1}c - A^{-1}b,$$

denn:

$$E_e^{-1}(E_e(p)) = E_e^{-1}(Ap + b) = A^{-1}(Ap + b) - A^{-1}b = A^{-1}Ap + A^{-1}b - A^{-1}b = p$$

für alle $p \in \mathbb{Z}_m^n$. Insbesondere ist $E_e^{-1} : \mathbb{Z}_m^n \to \mathbb{Z}_m^n$ affin linear. ◁

Beispiel 2.11 Eine affin lineare Verschlüsselungsfunktion auf $\mathbb{Z}_3$ mit Blocklänge 2 ist durch

$$\mathbb{Z}_3^2 \to \mathbb{Z}_3^2,\ p \mapsto Ap + b = \begin{pmatrix} [1] & [2] \\ [0] & [1] \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \begin{pmatrix} [1] \\ [1] \end{pmatrix}$$

gegeben. Die zu A inverse Matrix ist $\begin{pmatrix} [1] & [1] \\ [0] & [1] \end{pmatrix}$, und es gilt

$$A^{-1}b = \begin{pmatrix} [1] & [1] \\ [0] & [1] \end{pmatrix} \cdot \begin{pmatrix} [1] \\ [1] \end{pmatrix} = \begin{pmatrix} [2] \\ [1] \end{pmatrix}.$$

Damit ist die Entschlüsselungsfunktion gegeben durch

$$\mathbb{Z}_3^2 \to \mathbb{Z}_3^2,\ c \mapsto A^{-1}c + A^{-1}b = \begin{pmatrix} [1] & [1] \\ [0] & [1] \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} [2] \\ [1] \end{pmatrix}.$$

◁

Aufgabe 2.12

1. Modellieren Sie die Vigenère-Chiffre mit fester Schlüssellänge $n \in \mathbb{N}$ als affin lineare Blockchiffre.
2. Zeigen Sie, dass Permutationschiffren lineare Blockchiffren sind. ◁

Konstruktion 2.13 (Knacken affin linearer Chiffren) Affin lineare Blockchiffren können mittels eines Known-Plaintext-Angriffs gebrochen werden und sind damit unsicher. Ein erfolgreicher Angriff funktioniert wie folgt:

Wir nehmen an, dass eine Verschlüsselungsfunktion $E_e : \mathbb{Z}_m^n \to \mathbb{Z}_m^n$ fixiert wurde. Die zugehörige Verschlüsselungsfunktion ist affin linear und sei durch das Paar (A, b) mit $A \in \mathbb{Z}_m^{n \times n}$ und $b \in Z_m^n$ gegen, d. h.

$$E_e : \mathbb{Z}_m^n \to \mathbb{Z}_m^n,\ \ x \mapsto Ax + b.$$

Wir nehmen an, dass dem Angreifer $n+1$ Klartexte $p_1, \dots, p_{n+1}$ und die zugehörigen Chiffretexte $E_e(p_1) = c_1, \dots, E_e(p_{n+1}) = c_{n+1}$ bekannt sind. Er möchte das Paar (A, b) und damit die Entschlüsselungsfunktion $E_e : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, p \mapsto Ap + b$ bestimmen.

Wir betrachten die Matrizen

$$P = (p_1 - p_0, \dots, p_n - p_0) \text{ und } C = (c_1 - c_0, \dots, c_n - c_0).$$

Dann gilt $AP = C$, denn

$$c_i - c_0 = E_e(p_i) - E_e(p_0) = (Ap_i + b) - (Ap_0 + b) = A(p_i - p_0)$$

für alle $i = 1, \dots, n$.

Gilt nun noch, dass P invertierbar ist, dann sind $A = CP^{-1}$ und $b = c_0 - Ap_0$ bekannt.

Das Paar (A, b) (und somit die Verschlüsselungsfunktion $E_e(x) = Ax + b$) wurde aus $n + 1$ bekannten Paaren von Klar- und Chiffretexten bestimmt. ◁

Aufgabe 2.14 Das Alphabet einer linearen Blockchiffre mit Blocklänge 2 sei $\{A, \dots, Z\}$. Es sei außerdem bekannt, dass `HAND` in `FOOT` verschlüsselt und der ECB-Mode genutzt wird. Bestimmen Sie die Schlüsselmatrix $A \in \mathbb{Z}_{26}^{2\times 2}$. ◁

Die Konstruktion sicherer und effizienter Blockchiffren ist eine komplexe Aufgabe. Wie wir oben gesehen haben, darf die Blockchiffre insbesondere nicht affin linear sein. Von Shannon wurden zwei Konzepte vorgeschlagen, die die Sicherheit einer Blockchiffre erhöhen: *Konfusion* und *Diffusion*.

Anschaulich gesprochen ist die Konfusion einer Blockchiffre groß, wenn die Verteilung der Chiffretexte in einer so komplizierten Weise von der Verteilung der Klartexte abhängt, dass ein Angreifer diese Abhängigkeit nicht ausnutzen kann. Die Caesar-Chiffre hat eine sehr geringe Konfusion, da sich die Verteilung der Klartextbuchstaben direkt auf die Verteilung der Chiffretextbuchstaben überträgt. Die Diffusion einer Blockchiffre ist groß, wenn jeder einzelne Buchstabe des Klartexts und jeder einzelne Buchstabe des Schlüssels möglichst viele Buchstaben des Chiffretextes beeinflussen. Somit ist auch die Diffusion der Caesar-Chiffre äußerst gering.

Verschlüsselungsmodi

Es gibt mehrere Methoden, aus Blockchiffren der Länge $n \in \mathbb{N}$ Kryptosysteme zu konstruieren, mit denen man auch längere Texte – und nicht nur Blöcke der Länge n – verschlüsseln kann. Diese Methoden werden *Verschlüsselungsmodi* genannt:

Wir gehen von einer Blockchiffre der Länge $n \in \mathbb{N}$ auf dem Alphabet A aus und konstruieren ein symmetrisches Kryptosystem $(\mathscr{P}, \mathscr{C}, \mathscr{K}, \mathscr{E}, \mathscr{D})$ mit folgenden Eigenschaften: Das Alphabet des neuen Kryptosystems ist $B = A^n$, der Klartext- und Chiffreraum $\mathscr{P} = \mathscr{C} = B^*$, d. h. die Buchstaben des neuen Kryptosystems sind die Blöcke des alten. Wir nehmen an, dass A eine (additiv geschriebene) Gruppe ist.

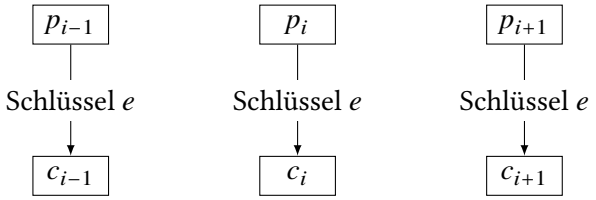


Abbildung 2.1.: Der ECB-Mode.

Beispiel 2.15 Das Alphabet der Blockchiffre sei $\mathbb{Z}_{26}$, die Blocklänge 3. Im neu konstruierten Kryptosystem ist

$$([3],[13],[4]) \in B = \mathbb{Z}_{26}^3$$

ein Buchstabe und

$$(([3],[13],[4]),([2],[21],[14]),([5],[1],[4]),([5],[19],[21])) \in (\mathbb{Z}_{26}^3)^4 \subset (\mathbb{Z}_{26}^3)^*$$

ein Wort. ◁

Bemerkung 2.16 Da $A^{n\cdot k}$ für $k \in \mathbb{N}$ als Gruppe isomorph zu $(A^n)^k = B^k \subset B^*$ ist, fassen wir auch Elemente von $A^{n\cdot k}$ als Elemente von $\mathcal{P} = \mathcal{C} = B^*$ auf. Für $A = \mathbb{Z}_{26}$ und $B = \mathbb{Z}_{26}^3$ gilt beispielsweise $A^{12} = \mathbb{Z}_{26}^{12} \cong (\mathbb{Z}_{26}^3)^4 = B^4$, und wir schreiben

$$([3],[13],[4],\ [2],[21],[14],\ [5],[1],[4],\ [5],[19],[21]) \in (\mathbb{Z}_{26}^3)^4 \subset (\mathbb{Z}_{26}^3)^*. \qquad \triangleleft$$

Für einen (geheimen) Schlüssel $e \in \mathcal{K}$ geben wir nun die zugehörige Verschlüsselungsfunktion $E_e : B^* \to B^*$ im neuen Kryptosystem an:

Sei $p_1p_2\ldots \in B^*$ ein Wort, d. h. $p_i \in B = A^n$ für alle i. Dann ist $E_e(p_1p_2\ldots) = c_1c_2\ldots$ wie folgt gegeben (wobei $c_i \in B = A^n$ für alle i); $c_0 \in B = A^n$ wurde im Vorhinein fest gewählt und ist öffentlich.

1. *ECB-Mode* (electronic codebook): $c_i := E_e(p_i)$ für alle $i \geq 1$
2. *CBC-Mode* (cipher-block chaining): $c_i := E_e(p_i + c_{i-1})$ für alle $i \geq 1$
3. *CFB-Mode* (cipher feedback): $c_i := p_i + E_e(c_{i-1})$ für alle $i \geq 1$
4. *OFB-Mode* (output feedback): $c_i := p_i + s_i$ für alle $i \geq 1$ mit $s_0 = c_0$ und $s_i = E_e(s_{i-1})$ für alle $i \geq 1$

Beispiel 2.17 (Verschlüsselungs-Modi) Als Blockchiffre wählen wir das One-Time-Pad mit Blocklänge 3. Der Schlüssel sei $e = ([10],[8],[4]) \in \mathbb{Z}_{26}^3$. Wir wählen $c_0 = ([1],[1],[1])$ und verschlüsseln den Klartext

$$p = ([12],[7],[23],\quad [2],[17],[4],\quad [1],[9],[13]) \in \mathbb{Z}_{26}^9$$

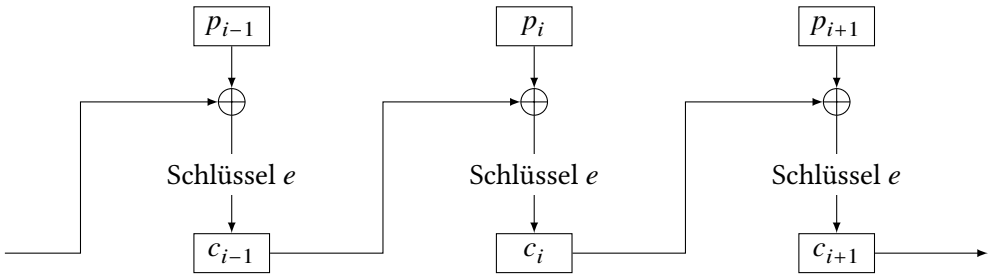


Abbildung 2.2.: Der CBC-Mode.

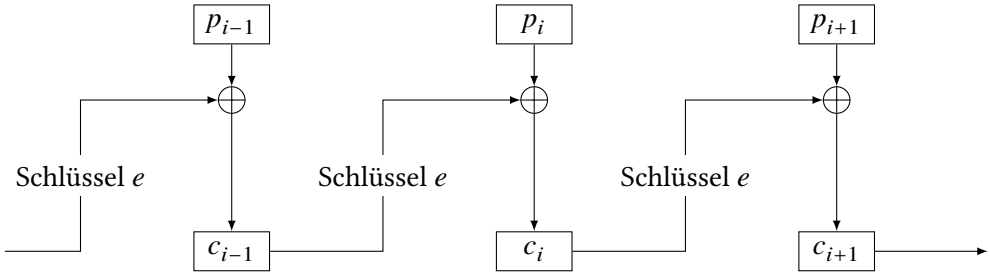


Abbildung 2.3.: Der CFB-Mode.

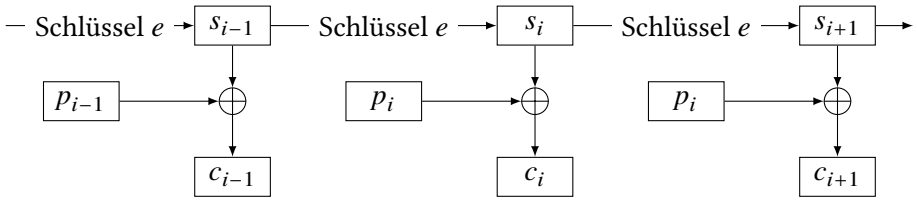


Abbildung 2.4.: Der OFB-Mode.

1. im ECB-Mode:

$$\begin{aligned} E_e(p) =&([12+10],[7+8],[23+4], \quad [2+10],[17+8],[4+4], \quad [1+10],[9+8],[13+4]) \\ =&([22],[15],[1], \quad [12],[25],[8], \quad [11],[17],[17]) \in \mathbb{Z}_{26}^9 \end{aligned}$$

Wir erhalten wieder die Vigenère-Chiffre mit Schlüssellänge 3, nun aber für Klartexte beliebiger Länge.

2. im CFB-Mode:

Da die Blocklänge 3 ist, erhalten wir

$$p_1 = ([12],[7],[23]),\ p_2 = ([2],[17],[4]) \text{ und } p_3 = ([1],[9],[13]).$$

Nun gilt

$$\begin{aligned} c_1 = p_1 + E_e(c_0) &= ([12],[7],[23]) + [([1],[1],[1]) + [10],[8],[4])] &&= ([23],[16],[2]), \\ c_2 = p_2 + E_e(c_1) &= ([2],[17],[4]) + [([23],[16],[2]) + [10],[8],[4])] &&= ([9],[15],[10]), \\ c_3 = p_3 + E_e(c_2) &= ([1],[9],[13]) + [([9],[15],[10]) + [10],[8],[4])] &&= ([20],[19],[27]). \end{aligned}$$

Insgesamt erhalten wir den Chiffretext

$$c = ([23],[16],[2], \quad [9],[15],[10], \quad [20],[19],[27]) \in \mathbb{Z}_{26}^9. \qquad \triangleleft$$

Aufgabe 2.18 1. Wir wählen den Schlüssel $e = ([1],[0],[1],[1]) \in \mathbb{Z}_2^4$, außerdem $c_0 = ([0],[0],[1],[1]) \in \mathbb{Z}_2^4$ und den Klartext

$$p = ([0],[1],[0],[0],\ [0],[1],[0],[0],\ [1],[1],[1],[0],\ [1],[0],[1],[1]) \in \mathbb{Z}_2^{16}.$$

Verschlüsseln Sie p mit den Verschlüsselungsmodi ECB-, CBC-, CFB- und OFB-Mode unter Nutzung des Vernam-One-Time-Pads als zugrunde liegender Blockchiffre mit Blocklänge 4.

Nehmen sie nun an, dass

$$c = ([1],[1],[0],[1],\ [1],[1],[0],[1],\ [0],[0],[1],[0],\ [1],[0],[0],[1]) \in \mathbb{Z}_2^{16}$$

mit e und dem ECB-, CBC-, CFB- bzw. OFB-Mode verschlüsselt wurde, wieder mit dem Vernam-One-Time-Pad als Blockchiffre. Bestimmen Sie jeweils den Klartext.

2. Implementieren Sie die Verschlüsselungsmodi ECB-, CBC-, CFB- und OFB-Mode für beliebige Blockchiffren. $\triangleleft$

2.2. AES

Der *Advanced Encryption Standard (AES)* ist ein flächendeckend genutztes Secret-Key-Kryptosystem. Der AES-Algorithmus ist frei verfügbar und darf ohne Lizenzgebühren eingesetzt werden. Er gilt bis heute als sicher. Die komplexeren Versionen von AES haben einen so guten Ruf, dass sie in den USA für staatliche Dokumente mit höchster Geheimhaltungsstufe zugelassen sind. Der erste theoretisch interessante Angriff wurde 2011 veröffentlicht. Dieser ist im Schnitt etwa um einen Faktor 4 schneller als ein vollständiges Durchsuchen des Schlüsselraums. Diese Beschleunigung ist so gering, dass AES trotz des Wissens um diesen theoretischen Angriff weiterhin als sicher gilt.

Das Alphabet, auf dem AES arbeitet, ist ein endlicher Körper mit 256 Elementen, der mit $\mathrm{GF}(2^8)$ bezeichnet wird. Dass $\mathrm{GF}(2^8)$ und nicht ein Ihnen bereits bekannter Körper $\mathbb{Z}_p$ mit einer Primzahl $p \in \mathbb{N}$ für AES genutzt wird, liegt an der Arbeitsweise von Computern:

Ein Verschlüsselungsalgorithmus soll schnell sein und wenig Speicher verbrauchen. Computer arbeiten auf Bits als kleinster Maßeinheit für Informationsgehalt. Ein Bit hat zwei Zustände, *Ein* oder *Aus*, die man in der Mathematik mit den Elementen von $\mathbb{Z}_2$ ausdrückt. Die nächstgrößere Maßeinheit sind Bytes, die aus 8 Bits bestehen. In der Mathematik ist ein Byte somit mit einem Element von $\mathbb{Z}_2^8$ identifiziert, z. B. stellt $([1],[0],[1],[1],[1],[0],[0],[1]) \in \mathbb{Z}_2^8$ ein spezielles Byte dar. Stehen die Elemente von $\mathbb{Z}_2^8$ für Bytes, nutzt man in der Regel die kürzere Notation 10111001.

Um eine effiziente Implementierung zu gewährleisten, ist es daher naheliegend für AES ein Alphabet mit 256 Buchstaben zu nutzen. Um auch komplexere Rechenoperationen, nämlich Divisionen auf dem Alphabet durchführen zu können, nutzt man die Struktur des Körpers $\mathrm{GF}(2^8) = \mathrm{GF}(256)$.

Es gibt ein weiteres Argument für ein Alphabet mit 256 Buchstaben (das allerdings auf dem ersten basiert): Unicode ist ein internationaler Standard, in dem langfristig für jedes Schriftzeichen (nicht nur das Standardalphabet, auch mathematische Symbole und arabische Zeichen) ein digitaler Code festgelegt wird. Die am weitesten verbreitete Codierung für Unicode-Zeichen ist UTF-8, eine Erweiterung von ASCII. Anfang 2015 nutzen über 80% der Webseiten UTF-8 zur Zeichencodierung. Die Buchstaben vieler westlicher Sprachen brauchen mit UTF-8 genau ein Byte Speicherplatz. Unter Nutzung von UTF-8 entsprechen damit häufig die Schriftzeichen im menschenlesbaren Text genau einem Buchstaben im Alphabet von AES. Beispielsweise wird das Wort „AES“ nach der Codierung mit UTF-8 zu

$$01000001 \quad 01000101 \quad 01010011.$$

Dies wäre die Nachricht im Klartextraum, die im Anschluss mithilfe des AES-Algorithmus verschlüsselt wird.

Obwohl AES auf dem endlichen Körper $\mathrm{GF}(2^8)$ als Alphabet arbeitet, verzichten wir in dieser in dieser Vorlesung auf die Konstruktion endlicher Körper. Interessierte Leser können die Ideen der Konstruktion in 2.3 nachlesen. Für unsere Zwecke ist der folgende Satz ausreichend, den wir ohne Beweis zitieren.

Satz 2.19 *Seien $n \in \mathbb{N}, p \in \mathbb{P}$. Dann existiert ein Körper mit p^n Elementen, der bis auf Isomorphie eindeutig ist. Wir bezeichnen diesen Körper mit* $\mathrm{GF}(p^n)$. ◁

Insbesondere existiert also ein Körper mit 2^8 Elementen, nämlich $\mathrm{GF}(2^8)$, der als Alphabet für AES genutzt wird.

Bemerkung 2.20 Die Formulierung „bis auf Isomorphie eindeutig“ bedeutet Folgendes: Sind K_1 und K_2 zwei Körper mit p^n Elementen, dann sind K_1 und K_2 isomorph. Zwei isomorphe Körper K_1 und K_2 unterscheiden sich anschaulich nur in die Bezeichnung ihrer Elemente, die Rechenregeln für + und · sind dieselben. ◁

Bemerkung 2.21 Das GF in $\mathrm{GF}(p^n)$ steht für „Galois field“: Évariste Galois war im 19. Jahrhundert ein französischer Mathematiker, der auf dem Gebiet der Körpertheorie gearbeitet hat und die nach ihm benannte Galoistheorie begründet hat. Er hat u. a. Körpererweiterungen untersucht, d. h. das Zusammenspiel von Körpern, die wie $\mathbb{R}$ in $\mathbb{C}$ ineinander enthalten sind. „field“ ist das englische Wort für Körper. ◁

Notation 2.22 (Elemente des Körpers $\mathbf{GF}(2^8)$) Obwohl wir den Körper $\mathrm{GF}(2^8)$ nicht formal einführen, möchten wir Rechenoperationen darauf beschreiben, um AES vorstellen zu können. Wir benutzen folgende Bezeichnung für die Elemente von $\mathrm{GF}(2^8)$:

1. Elemente von $\mathbb{Z}_2^8$, z. B. $([0],[1],[0],[1],[0],[0],[1],[0]) \in \mathrm{GF}(2^8)$
2. zweistellige Hexadezimalzahlen, z. B. `d2`

Wie üblich bezeichnen wir die Ziffern im Hexadezimalsystem, dem Stellenwertsystem zur Basis 16, mit

`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.`

Die Zahlen `c` und `4c` im Hexadezimalsystem entsprechen beispielsweise den Zahlen 12 und $4 \cdot 16 + 12 \cdot 1 = 76$ im Dezimalsystem.

Die Umrechnung eines Elements in $\mathbb{Z}_2^8$ in eine zweistellig Hexadezimalzahl funktioniert beispielhaft wie folgt:

Der Vektor $([1],[0],[1],[1],[1],[0],[0],[1]) \in \mathbb{Z}_2^8$ wird als die Binärzahl 10111001 aufgefasst. Bei der Umwandlung ins Hexadezimalsystem gehen die ersten vier Stellen von 10111001 auf die erste Stelle im Hexadezimalsystem, die letzten vier Stellen auf die zweite Stelle. 1011 im Binärsystem ist 11 im Dezimalsystem, also `b` im Hexadezimalsystem. 1001 im Binärsystem ist 9 im Dezimalsystem, also `9` im Hexadezimalsystem. Insgesamt erhalten wir `b9` als Hexadezimaldarstellung des Vektors $([1],[0],[1],[1],[1],[0],[0],[1]) \in \mathbb{Z}_2^8$.

Die Addition auf $\mathrm{GF}(2^8)$ entspricht der Addition auf $\mathbb{Z}_2^8$, die Multiplikation auf $\mathrm{GF}(2^8)$ führen wir nicht ein. ◁

Von endlichen Körpern zurück zu AES:

Der AES-Algorithmus ist ein Spezialfall der sogenannten *Rijndael-Chiffre*, die von den Belgiern Joan Daemen und Vincent Rijmen in den 90-ern entwickelt wurde. Es gibt mehrere Versionen des AES-Algorithmus, die sich durch die Schlüssellänge $\mathrm{Nk} = 16, 24, 32$ und die Anzahl der Runden

$$\mathrm{Nr} = \begin{cases} 10 & \text{für } \mathrm{Nk} = 16, \\ 12 & \text{für } \mathrm{Nk} = 24, \\ 14 & \text{für } \mathrm{Nk} = 32 \end{cases}$$

unterscheiden. Um die Darstellung nicht unnötig kompliziert werden zu lassen, beschränken wir uns in dieser Vorlesung auf die Schlüssellänge $\mathrm{Nk} = 16$ und damit die Rundenanzahl $\mathrm{Nr} = 10$.

Wir modellieren AES als Blockchiffre:

Das Alphabet von AES ist $A = \mathrm{GF}(2^8)$, die Blocklänge ist $n = 16$, damit gilt für den Klartext- und den Chiffretextraum

$$\mathscr{P} = \mathscr{C} = \mathrm{GF}(2^8)^{16};$$

ein Beispiel für einen Klartext (in Hexadezimaldarstellung) ist

$$\texttt{32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34} \in \mathscr{P}.$$

Der Schlüsselraum der hier beschriebenen Version von AES ist ebenfalls $\mathscr{K} = \mathrm{GF}(2^8)^{16}$.

Für einen Schlüssel $e \in \mathscr{K}$, z. B.

$$\texttt{3d c3 16 a3 28 ba a0 fd 3e 32 92 52 e0 b7 0b b4} \in \mathscr{K} = \mathrm{GF}(2^8)^{16},$$

beschreiben wir nun den Verschlüsselungsalgorithmus $E_e : \mathscr{P} \to \mathscr{C}$:

Aus dem Verschlüsselungsschlüssel $e \in \mathrm{GF}(2^8)^{16}$ wird ein expandierter Schlüssel

$$\texttt{KeyExpansion}(e) = (e_1, \dots, e_{10}) \quad \in (\mathrm{GF}(2^8)^{16})^{10} = \mathscr{P}^{10} = \mathscr{C}^{10}$$

erzeugt, z. B.

```
29 ee cb 34 44 85 93 3a 5b d6 5c 47 9f 11 3a 70
32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
49 8f 52 43 1f e1 cb e2 bf 94 3c d0 30 f5 55 a0
6e c2 63 7f 27 3b ee 7f f0 2f 74 ac 92 21 22 c6
3b b9 73 ae c9 f1 7c 78 66 60 86 6e 85 9d 04 70
ec 1a 11 21 0e 6b 86 1a a6 70 d5 04 a8 fc ff 5d
0f de 2f aa 20 ce 47 5f da 09 42 22 b6 cc 2f ba
db 7a 6a 30 7f e5 e2 07 52 94 24 4d 7c 19 62 37
a8 fe c0 b7 58 16 8e 2c 4a de 6e 7c 8e eb 24 4a
f0 34 ab 36 84 89 06 4e 29 3a 6b 0f 82 09 72 be
```
$\in \mathscr{K}^{10} = (\mathrm{GF}(2^8)^{16})^{10}$.

Auf den verwendeten Algorithmus gehen wir in dieser Vorlesung nicht ein, stattdessen gehen wir davon aus, dass der expandierte Schlüssel $(e_1, \dots, e_{10})$ bereits vorliegt.

Die Verschlüsselungsfunktion E_e durchläuft $\mathrm{Nr} = 10$ Runden, in denen jeweils die Transformationen `SubBytes`, `MixColumns`, `ShiftRows` und eine Addition erfolgen. Zusätzlich erfolgt eine Vor- und eine Nachbearbeitung. Die genauen Schritte, die E_e zugrunde liegen, sind in Algorithmus 2.2 aufgeführt. Wir beschreiben nun die Transformationen `SubBytes`, `MixColumns` und `ShiftRows`.

Algorithmus 1 Advanced Encryption Standard (AES)

Input: Klartext $p \in \mathcal{P} = \mathrm{GF}(2^8)^{16}$, Schlüssel $e \in \mathcal{K} = \mathrm{GF}(2^8)^{16}$
Output: Chiffretext $c = E_e(p) \in \mathcal{C} = \mathrm{GF}(2^8)^{16}$

```
(e_1, ..., e_10) ← KeyExpansion(e)
c ← p + e
for i = 1, ..., 9 do
    c ← SubBytes(c)
    c ← ShiftRows(c)
    c ← MixColumns(c)
    c ← c + e_i
end for
c ← SubBytes(c)
c ← ShiftRows(c)
c ← c + e_10
return c
```

Die Transformation `SubBytes`

Die Transformation `SubBytes` arbeitet buchstabenweise durch Verwürfelung der Buchstaben des Alphabets $\mathrm{GF}(2^8)$. Wir definieren die Abbildung

$$\sigma : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8), \quad b \mapsto A \cdot b^{-1} + c$$

durch

$$A = \begin{pmatrix} [1] & [0] & [0] & [0] & [1] & [1] & [1] & [1] \\ [1] & [1] & [0] & [0] & [0] & [1] & [1] & [1] \\ [1] & [1] & [1] & [0] & [0] & [0] & [1] & [1] \\ [1] & [1] & [1] & [1] & [0] & [0] & [0] & [1] \\ [1] & [1] & [1] & [1] & [1] & [0] & [0] & [0] \\ [0] & [1] & [1] & [1] & [1] & [1] & [0] & [0] \\ [0] & [0] & [1] & [1] & [1] & [1] & [1] & [0] \\ [0] & [0] & [0] & [1] & [1] & [1] & [1] & [1] \end{pmatrix} \in \mathbb{Z}_2^{8\times 8} \text{ und } c = \begin{pmatrix} [1] \\ [1] \\ [0] \\ [0] \\ [0] \\ [1] \\ [1] \\ [0] \end{pmatrix}.$$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Abbildung 2.5.: Tabellierung von `SubBytes`.

Man kann zeigen, dass $\det A = [1] \in \mathbb{Z}_2$ gilt und dass A somit invertierbar ist. Die Abbildung σ ist nicht affin linear (man beachte das b^{-1}), was AES vor dem in Abschnitt 2.1 beschriebenen Known-Plaintext-Angriff auf affin lineare Blockchiffren schützt.

Die Permutation $\sigma : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8)$, über die `SubBytes` definiert ist, kann wie in Abbildung 2.5 tabelliert werden. Beispielsweise kann dort $\sigma(\texttt{1d}) = \texttt{a4}$ abgelesen werden.

Die Transformation `SubBytes` wendet die Abbildung σ buchstabenweise an:

$$\texttt{SubBytes}(p_1, \dots, p_{16}) = (\sigma(p_1), \dots, \sigma(p_{16}))$$

für alle $(p_1, \dots, p_{16}) \in \mathrm{GF}(2^8)^{16} = \mathcal{P} = \mathcal{C}$.

Beispielsweise gilt

$$\begin{aligned}\texttt{SubBytes(32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34)}\\ = \texttt{23 1a 42 c2 c4 be 04 5d c7 c7 46 3a e1 9a c5 18.}\end{aligned}$$

Die Transformation `ShiftRows`

Die Transformation `ShiftRows` permutiert die Buchstaben eines Blocks $p = (p_1, \dots, p_{16}) \in \mathrm{GF}(2^8)^{16}$:

$$\texttt{SubBytes}(p) = (p_1, p_6, p_{11}, p_{16}, p_5, p_{10}, p_{15}, p_4, p_9, p_{14}, p_3, p_8, p_{13}, p_2, p_7, p_{12})$$

Beispielsweise gilt:

```
ShiftRows(23 1a 42 c2 c4 be 04 5d c7 c7 46 3a e1 9a c5 18)
        = 23 be 46 18 c4 c7 c5 c2 c7 9a 42 5d e1 1a 04 3a
----------------------------------------------------------
          01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
```

Die Transformation `MixColumns`

Die Transformation `MixColumns` operiert auf Blöcken der Länge 4, also auf $\mathrm{GF}(2^8)^4$. Wir definieren die lineare Abbildung

$$S : \mathrm{GF}(2^8)^4 \to \mathrm{GF}(2^8)^4, \quad \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} \mapsto \begin{pmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix}.$$

Die Einträge der Matrix sind Hexadezimalzahlen, die für die entsprechenden Elemente von $\mathrm{GF}(2^8)$ stehen.

Die Transformation `MixColumns` wendet die $\mathrm{GF}(2^8)$-lineare Abbildung S blockweise an:

$$\texttt{MixColumns}(p) = (S(p_1, p_2, p_3, p_4), S(p_5, p_6, p_7, p_8), S(p_9, p_{10}, p_{11}, p_{12}), S(p_{13}, p_{14}, p_{15}, p_{16}))$$

für alle $(p_1, \dots, p_{16}) \in \mathrm{GF}(2^8)^{16} = \mathscr{P} = \mathscr{C}$.

Da die Multiplikation auf $\mathrm{GF}(2^8)$ nicht eingeführt wurde, kann kein Beispiel für die Abbildung S und die Transformation `MixColumns` angegeben werden.

Dekodierung

Um einen mit AES verschlüsselten Text zu dechiffrieren, muss man den AES-Algorithmus invertieren. Dazu geht man den Algorithmus von hinten nach vorne durch und kehrt die durchgeführten Schritte um. Dies ist möglich, da alle genutzten Transformationen `SubBytes`, `MixColumns`, `ShiftRows` und auch die Addition invertierbar sind:

- `SubBytes` arbeitet buchstabenweise durch (bijektive) Verwürfelung des Alphabets. Diese Verwürfelung kann umgekehrt werden.

- `ShiftRows` permutiert die Buchstaben eines Textes, ist also bijektiv. Die Permutation der Buchstaben kann umgekehrt werden.
- `MixColumns` arbeitet blockweise als invertierbare lineare Abbildung $S : \mathrm{GF}(2^8)^4 \to \mathrm{GF}(2^8)^4$. Es existiert also eine (blockweise arbeitende) inverse lineare Abbildung.

Kryptoanalyse

Eine Kryptoanalyse von AES ist schwierig und aufwendig. Das liegt nicht daran, dass die einzelnen Schritte komplex und undurchsichtig sind, sondern daran, dass sehr viele Schritte während des Algorithmus durchlaufen werden. Erst durch das Aneinanderreihen der einzelnen Transformation in mehreren Runden wird AES komplex und undurchsichtig.

Die einzelnen durchgeführten Schritte haben sogar eine sehr einfach Struktur: Bis auf `SubBytes` sind alle Transformationen affin linear und wären für sich allein genommen zu Knacken. Daher ist der Einsatz von `SubBytes` so wichtig, um AES vor dem in Kapitel 2.1 beschriebenen Known-Plaintext-Angriff zu schützen.

Gerade diese Einfachheit der einzelnen Transformationen ist wiederum eine Stärke von AES: Sie garantiert, dass AES effizient arbeitet, d. h. wenig Speicher und wenig Rechenzeit benötigt. Neben der Sicherheit des Algorithmus war das ein Hauptkriterium bei der Auswahl von AES im 1997 vom NIST ausgeschriebenen Wettbewerb.

2.3. Exkurs: Endliche Körper

Für interessierte Leser beschreiben wir aufbauend auf [Gat10] die Konstruktion endlicher Körper. Dazu führen wir zuerst den Polynomring und Teilbarkeitsbegriffe ein. Im Folgenden sei ein Ring stets kommutativ und mit Eins.

Der Polynomring

Definition 2.23 (Polynome) Sei R ein Ring.

1. Ein *Polynom in einer Variablen über R* ist ein Ausdruck der Form

$$a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0$$

mit *Koeffizienten* $a_n, \ldots, a_0 \in R$ und $n \in \mathbb{N}$. Wir fassen dabei t als *formale Variable* auf. Natürlich könnte man hier auch einen anderen Buchstaben als formale Variable wählen, in diesem Exkurs werden wir die formale Variable eines Polynoms jedoch immer mit t bezeichnen.

Wir definieren $R[t]$ als die Menge aller Polynome über R.

2. Sind $f = \sum_{k=0}^{n_f} a_k t^k$ und $g = \sum_{k_0}^{n_g} b_k t^k$ Polynome, so definieren wir ihre Summe $f + g$ und ihr Produkt $f \cdot g$ als die Polynome

$$\begin{aligned} f + g &:= \sum_{k=0}^{n_f} (a_k + b_k) t^k \\ \text{und} \quad f \cdot g &:= \sum_{m=0}^{n_f+n_g} \left(\sum_{k+l=m} (a_k + b_l) \right) t^m. \end{aligned} \tag{2.1}$$

◁

Bemerkung 2.24 Man beachte, dass die Multiplikation von Polynomen genauso eingeführt wurde, wie man es erwarten würde, wenn man diese formalen Summen naiv ausmultiplizieren würde: dann wäre nämlich

$$\left(\sum_{k=0}^{n_f} a_k t^k \right) \cdot \left(\sum_{l=0}^{n_g} b_l t^l \right) = (a_0 + a_1 t + a_2 t^2 + \dots) \cdot (b_0 + b_1 t + b_2 t^2 + \dots) \tag{2.2}$$

$$= a_0 b_0 + (a_0 b_1 + a_1 b_0) t + (a_0 b_2 + a_1 b_1 + a_2 b_0) t^2 + \dots \tag{2.3}$$

$$= \sum_{m=0}^{n_f+n_g} \left(\sum_{k+l=m} (a_k + b_l) \right) t^m, \tag{2.4}$$

was genau die obige Definition ist. ◁

Beispiel 2.25 Über einem Ring R betrachten wir das Polynom

$$f = 1 + 1t + 1t^2 + \dots + 1t^n \in R[t].$$

Dann ist nach 2.23 die Summe von f mit sich selbst

$$f + f = 2f = \sum_{k=0}^{n} (1 + 1) t^n = \sum_{k=0}^{n} 2t^n$$

und das Produkt von f mit sich selbst

$$f \cdot f = f^2 = \sum_{m=0}^{2n} \left(\sum_{k+l=m} 1 \cdot 1 \right) t^n = \sum_{m=0}^{2n} (m + 1) t^n,$$

da die Summe über k und l aus $m + 1$ Summanden besteht. ◁

Notation 2.26 Aus naheliegenden Gründen lässt man Monome von der Form $0t^n$ bei der Notation von Polynomen häufig weg und schreibt t^n für $1t^n$. ◁

Satz 2.27 *Für jeden Körper R ist die Menge $R[t]$ aller Polynome zusammen mit den Verknüpfungen aus Definition 2.23 ein Ring. R ist ein Unterring von $R[t]$, insbesondere sind $0 \in R$ und $1 \in R$ die neutralen Elemente in $R[t]$ bezüglich Addition und Multiplikation.* ◁

Definition 2.28 (Grad, Leitkoeffizient) Wieder sei R ein Ring.

1. Ist $f \in R[t]$ ein Polynom mit $f \neq 0$, so können wir f eindeutig als

$$f = a_0 + a_1 t + \ldots + a_n t^n$$

mit $n \in \mathbb{N}$ und $a_n \neq 0$ schreiben. Die Zahl n, also der höchste in f auftretende Exponent von t wird der *Grad von* f genannt und als $\deg f$ geschrieben (die Bezeichnung kommt vom englischen Wort „degree“). Dann heißt $a_n \in R$ Leitkoeffizient von f.

2. Ein Polynom $a_0 \in R \subset R[t]$ vom Grad 0 heißt *konstantes Polynom.* ◁

Definition 2.29 (Polynomfunktion) Ist $f = a_0 + a_1 t + \ldots + a_n t^n$ ein Polynom über dem Körper K und ist $x \in R$, so heißt das Ringelement

$$f(x) := a_0 + a_1 x + \ldots + a_n x^n \quad \in R$$

der *Wert von* f *in* x. die zugehörige Funktion $R \to R$, $x \mapsto f(x)$, wird eine *Polynomfunktion* genannt. Ist $f(x) = 0 \in R$, so heißt $x \in R$ eine *Nullstelle von* f. ◁

Beispiel 2.30 Jedes Polynom f über R bestimmt eine Polynomfunktion. Zwei verschiedene Polynome können dabei dieselbe Polynomfunktion bestimmen: Das Polynom $t^2 + t \in \mathbb{Z}_2[x]$ hat in jedem Element von $\mathbb{Z}_2$ den Wert Null. Während die beiden *Polynome* $t^2 + t \in \mathbb{Z}_2[t]$ und $[0] \in \mathbb{Z}_2[t]$ verschieden sind, sind die beiden zugehörigen Polynomfunktionen $t \mapsto t^2 + t$ und $t \mapsto [0]$ gleich. Man beachte, dass man deshalb einer *Polynomfunktion* in der Regel keinen eindeutigen Grad zuweisen kann. Polynome verhalten sich hier also besser als Polynomfunktionen.

Man kann allerdings zeigen, dass Polynome und Polynomfunktionen über Körpern mit unendlich vielen Elementen (also z. B. $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$) übereinstimmen. In diesem Fall macht es also keinen Unterschied, ob wir bei Polynomen an Funktionen oder an die formalen Ausdrücke aus Definition 2.23 denken. ◁

Teilbarkeit im Ringen

Definition 2.31 (Teilbarkeit) Sei R ein kommutativer Ring mit 1 und seien $a, b \in R$.

- Man sagt a *teilt* b und schreibt $a \mid b$, wenn ein $c \in R$ existiert, das $a \cdot c = b$ erfüllt.
- Das Element a heißt *Einheit*, wenn ein $c \in R$ existiert mit $a \cdot c = 1 \in R$. Die Menge aller Einheiten von R wird mit R^* bezeichnet und heißt *Einheitengruppe von* R.
- Das Element a heißt *Nullteiler*, wenn ein $c \in R$ existiert mit $a \cdot c = 0 \in R$.
- Der Ring R heißt *Integritätsring*, wenn er außer der Null keine Nullteiler besitzt, d. h. wenn für alle $x, y \in R$ gilt: Ist $ab = 0$, so ist bereits $a = 0$ oder $b = 0$. ◁

Beispiel 2.32 Ein Körper K ist ein Integritätsring: Angenommen es würden Elemente $0 \neq a, b \in K$ existieren, die $a \cdot b = 0$ erfüllen. Dann hätte a ein Inverses a^{-1} bezüglich der Multiplikation, und wir erhalten den Widerspruch $a^{-1} \cdot a \cdot b = b = 0$. ◁

Aufgabe 2.33 Sei R ein Integritätsring. Zeigen Sie:

1. Sind $0 \neq f, g \in R[t]$ Polynome über R, so gilt
$$\deg(f+g) \leq \max\{\deg f, \deg g\} \text{ und } \deg(f \cdot g) = \deg f + \deg g.$$
2. Es existiert ein Ring R und Polynome $0 \neq f, g \in R[t]$ mit $\deg(f \cdot g) < \deg f + \deg g$. Ein Tipp: Der Ring R kann in diesem Fall kein Integritätsring, also insbesondere kein Körper sein. ◁

Aufgabe 2.34 (Einheitengruppe des Polynomrings) Sei R ein Integritätsring. Zeigen Sie, dass die Einheitengruppe des Polynomrings über R durch $R[t]^* = R^*$ gegeben ist. ◁

Definition 2.35 (Primelemente und irreduzible Elemente) Sei R ein Integritätsring und $p \in R$ mit $p \neq 0$ und $p \neq R^*$.

1. p heißt *irreduzibel*, wenn für alle $a, b \in R$ mit $p = a \cdot b$ gilt, dass $a \in R^*$ oder $b \in R^*$ eine Einheit ist. Andernfalls heißt p *reduzibel*.
2. p heißt *prim*, wenn für alle $a, b \in R$ mit $p \mid a \cdot b$ gilt, dass $p \mid a$ oder $p \mid b$. ◁

Lemma 2.36 *In einem Integritätsring ist jedes Primelement irreduzibel.* ◁

Bemerkung 2.37 Die Umkehrung dieses Lemmas gilt nicht: Es gibt Integritätsringe, in denen nicht alle irreduziblen Elemente prim sind. ◁

Teilbarkeit im Polynomring

Bezüglich der Teilbarkeit ist in Polynomringen vieles einfacher als in allgemeinen Ringen. Das liegt im Wesentlichen daran, dass man Polynome mit Rest durcheinander teilen kann. Das Verfahren der Division mit Rest sollte aus der Geometrie-Vorlesung bekannt sein.

Satz 2.38 (Polynomdivision) *Sind $f, g \in K[t]$ Polynome mit $g \neq 0$, so gibt es stets Polynome $q, r \in K[t]$ mit $f = qg + r$ und $\deg r < \deg g$.* ◁

Satz 2.39 *Sei K ein Körper und $0 \neq f \in K[t]$ ein Polynom mit $f \notin K[t]^*$. Dann hat f eine* eindeutige Primfaktorzerlegung*, d. h.:*

1. *Es gibt ein $n \in \mathbb{N}$ und Primelemente $p_1, \ldots, p_n \in R$, so dass $a = p_1 \cdot \ldots \cdot p_n$ gilt. Eine solche Darstellung heißt* Primfaktorzerlegung von f.
2. *Die Darstellung aus 1. ist „bis auf Reihenfolge und bis auf Multiplikation mit Einheiten eindeutig“.* ◁

Lemma 2.40 *Sei K ein Körper und $f \in K[t]$. Dann ist f genau dann prim, wenn es irreduzibel ist.* ◁

Aufgabe 2.41 Sei K ein Körper und $p = t - a \in K[t]$ ein konstantes Polynome. Man zeige, dass f dann irreduzibel ist. ◁

Korollar 2.42 (Abspalten von Nullstellen in Polynomen) *Es sei K ein Körper und $0 \neq f \in K[t]$ ein Polynom vom Grad $n \in \mathbb{N}_{\geq 0}$.*

1. *Ist $a \in K$ eine Nullstelle von f, so gilt $x - a \mid f$.*
2. *Das Polynom f hat höchstens n Nullstellen.* ◁

Beispiel 2.43

1. Wir zeigen, dass $f = t^2 + t + 1 \in \mathbb{Z}_2[t]$ irreduzibel ist: Wäre f reduzibel, so müsste f nach Aufgabe 2.33 Produkt von zwei Polynomen vom Grad 1 sein. Dann hätte f eine Nullstelle in $\mathbb{Z}_2$. Es gilt aber $f([0]) = f([1]) = [1] \in \mathbb{Z}_2$.
2. Das Polynom $f = t^2 + [1]$ ist reduzibel in $\mathbb{Z}_2[t]$: Es gilt $(t + [1])(t + [1]) = t^2 + [2]t + [1] \in \mathbb{Z}_2[t]$, wobei $t + [1] \in \mathbb{Z}_2[t]$ gemäß 2.34 keine Einheit ist. ◁

Bemerkung 2.44 (Größter gemeinsamer Teiler) Analog zu $\mathbb{Z}$ sind im Polynomring $K[t]$ über einem Körper K *größte gemeinsame Teiler* $\mathrm{ggT}(f, g)$ *zweier Polynome* f, g definiert:

Ein Polynom $d \in K[t]$ heißt größter gemeinsamer Teiler von f und g, wenn d gemeinsamer Teiler von f und g ist und wenn für alle gemeinsamen Teiler h von f und g auch $h \mid d$ gilt. Man kann zeigen, dass zwei Polynome f, g immer größte gemeinsame Teiler besitzen. In der Regel sind diese allerdings nicht eindeutig. Wie im Fall von $\mathbb{Z}$ kann ein größter gemeinsamer Teiler d von f und g mithilfe des aus der Geometrie-Vorlesung bekannten erweiterten euklidischen Algorithmus bestimmt und als Linearkombination

$$d = a_1 f + a_2 g$$

von f und g über $K[t]$ dargestellt werden, d. h. $a_1, a_2 \in K[t]$. ◁

Aufgabe 2.45 Bestimmen Sie mithilfe des erweiterten euklidischen Algorithmus einen größten gemeinsamen Teiler von $t^3 + [1] \in \mathbb{Z}_2[t]$ und $t^3 + t^2 + t \in \mathbb{Z}_2[t]$. Stellen sie diesen Teiler als Linearkombination von f und g über $K[t]$ dar. ◁

Konstruktion endlicher Körper

Die Elemente des endlichen Körpers, der nun konstruiert wird, sind Restklassen in $\mathbb{Z}_p[t]$ modulo eines Polynoms f über $\mathbb{Z}_p$. Die Konstruktion dieser Restklassen entspricht der Konstruktion von Restklassen in $\mathbb{Z}$ modulo $n \in \mathbb{N}$.

Definition 2.46 (Restklassenring modulo (f)) Sei K ein Körper und $f, g \in K[t]$. Wir bezeichnen mit $(f) := f \cdot K[t] := \{f \cdot h \mid h \in K[t]\}$ die Menge aller Vielfachen von f und nennen (f) das *Erzeugnis von* f. Die *Restklasse in* $K[t]$ *von* g *modulo* f ist definiert als

$$g + (f) = g + f \cdot K[t] := \{g + f \cdot h \mid h \in K[t]\}.$$

Sei $h \in g + (f)$ ein Polynom. Dann nennen wir h einen *Repräsentanten von* $g + (f)$.

Die Menge aller Restklassen in $K[t]$ modulo f bezeichnen wir mit $K[t]/(f)$ und nennen sie den *Restklassenring von* $K[t]$ *modulo* f. ◁

Satz 2.47 *Sei K ein Körper und $f \in K[t]$. Dann bildet die Menge der Restklassen $K[t]/(f)$ mit folgenden Operationen einen Ring $(K[t], +, \cdot)$:*

$$(g_1 + (f)) + (g_2 + (f)) := g_1 + g_2 + (f) \tag{2.5}$$
$$(g_1 + (f)) \cdot (g_2 + (f)) := g_1 \cdot g_2 + (f) \tag{2.6}$$

Ist f irreduzibel, dann ist $K[t]/(f)$ sogar ein Körper. ◁

Bemerkung 2.48 Insbesondere ist $0 + (f) = (f)$ das neutrale Element der Addition in $K[t]/(f)$ und $1 + (f)$ das neutrale Element der Multiplikation. Damit ist $-g_1 + (f)$ das zu $g_1 + (f)$ inverse Element bezüglich $+$. ◁

Bemerkung 2.49 Man beachte, dass „+“ in Satz 2.47 für drei verschiedenen Operationen steht: für die Addition auf $K[t]$, für die Addition auf $K[t]/(f)$ und für die Summe von einem Polynom $g \in K[t]$ und der Menge von Polynomen (f). Da alle diese Operationen auf die Addition in $K[t]$ zurück geführt werden, ist diese Vereinheitlichung der Schreibweise sinnvoll und praktisch. Analog steht $\cdot$ für zwei verschiedene Operationen: für die Multiplikation auf $K[t]$ und die auf $K[t]/(f)$. ◁

Beispiel 2.50 Wir betrachten den Körper $K = \mathbb{Z}_2$ und das Polynom $f = t$.

Die Restklasse $[0] + (t) = \{[0] + t \cdot h \mid h \in \mathbb{Z}_2[t]\}$ enthält alle Polynome, die ein Vielfaches von t sind, z. B. gilt

$$t^5 + t^3 = [0] + t \cdot (t^4 + t^2) \in [0] + (t).$$

Dahingegen enthält $[1] + (t) = \{[1] + t \cdot h \mid h \in K[t]\}$ alle Polynome, die sich um $[1]$ von einem Vielfachen von t unterscheiden.

Insgesamt gilt $\mathbb{Z}_2[t]/(t) = \{f + (t) \mid f \in K[t]\} = \{[0] + (t), [1] + (t)\}$:

Jedes $g = a_0 + a_1 t + \ldots + a_n t^n \in Z_2[t]$ ist wegen $a_0 \in \{[0], [1]\} = \mathbb{Z}_2$ entweder ein Vielfaches von t oder unterscheidet sich um $[1]$ von einem Vielfachen von t.

Da t irreduzibel ist, ist $\mathbb{Z}_2[t]/(t)$ außerdem ein Körper. Er ist isomorph zu $\mathbb{Z}_2$. ◁

Wir gehen nun der Frage nach, was die Menge der Restklassen modulo f anschaulich darstellt. Dazu stellen wir einige Rechenregeln für die oben definierten Restklassen auf:

Lemma 2.51 *Seien K ein Körper und $f, g_1, g_2 \in K[t]$. Dann gilt:*

1. $(a) = K[t]$ *für alle konstanten Polynome* $a \in K^*$
2. $g_1 + K[t] = K[t]$
3. $g_1 + (f) = 0 + (f) \quad \Leftrightarrow \quad g_1 \in (f)$
4. $g_1 + (f) = g_2 + (f) \quad \Leftrightarrow \quad g_1 - g_2 \in (f)$ ◁

Beweis.

1. Die Abbildung $K[t] \to K[t], h \mapsto a \cdot h$ ist bijektiv mit Bild $K[t] = \{a \cdot h \mid h \in K[t]\} = (a)$.
2. Die Abbildung $K[t] \to K[t], h \mapsto g + h$ ist bijektiv mit Bild $K[t] = g + K[t]$.
3. „⇒“: Es gelte $g_1 + (f) = 0 + (f)$, also $\{g_1 + f \cdot h \mid h \in K[t]\} = \{f \cdot h \mid h \in K[t]\}$. Insbesondere gilt dann
$$g_1 = g_1 + f \cdot 0 \in \{f \cdot h \mid h \in K[t]\} = (f).$$
„⇐“ Wegen $g_1 \in (f) = \{f \cdot h \mid h \in K[t]\}$ existiert ein $h_1 \in K[t]$ mit $g_1 = f \cdot h_1$. Wegen der 2. Eigenschaft gilt $\{h_1 + h \mid h \in K[t]\} = \{h \mid h \in K[t]\}$, und wir folgern
$$g_1 + (f) = \{f \cdot h_1 + f \cdot h \mid h \in K[t]\} = \{f \cdot (h' + h) \mid h \in K[t]\} = \{f \cdot h \mid h \in K[t]\} = (f).$$
4. „⇒“: Es gelte $g_1 + (f) = g_2 + (f)$. Daraus folgt
$$0 + (f) = (-g_2 + (f)) + (g_1 + (f)) = -g_2 + g_1 + (f).$$
Mit 3. erhalten wir $g_1 - g_2 \in (f)$.
„⇐“: Aus $g' := g_1 - g_2 \in (f)$ folgern wir mit der 3. Eigenschaft
$$g_1 + (f) = g_2 + g' + (f) = g_2 + (f).$$

□

Nun sind wir in der Lage zwei sich ergänzende Anschauungen des Restklassenrings zu gewinnen.

Zuerst einmal stellen wir mit Satz 2.47 fest, dass wir im Restklassenring im Prinzip wie im Polynomring rechnen können: zwei Restklassen $g_1 + (f)$ und $g_2 + (f)$ werden addiert, indem man ihre Repräsentanten zu $g_1 + g_2$ addiert und dann die zugehörige Restklasse $g_1 + g_2 + (f)$ berechnet; genauso für die Multiplikation.

Zum anderen gibt es Unterschiede zum Polynomring:

1. Betrachten wir nochmals die 3. Aussage in Lemma 2.51. Diese besagt, dass die Restklassen $g + (f)$ aller Polynome $g \in (f)$ gleich sind, nämlich gleich $0 + (f)$. Anschaulich gesprochen werden alle Elemente $g \in (f)$ im Restklassenring $K[t]/(f)$ zur Null. Man kann im Restklassenring also so rechnen wie im Polynomring, mit der zusätzlich Eigenschaft, dass alle Elemente von (f) (sprich: alle Vielfachen von f) zu Null geworden sind.

2. Betrachten wir nun die 4. Aussage in Lemma 2.51: Zwei Restklassen $g_1 + (f)$ und $g_2 + (f)$ sind genau dann gleich, wenn ihre Differenz $g_1 - g_2$ ein Element von (f), also ein Vielfaches von f ist.

   Man kann mit Restklassen also rechnen wie mit Polynomen, mit der zusätzlichen Eigenschaft, dass zwei Elemente gleich sind, wenn sie sich nur um ein Element von (f) unterscheiden.

3. Diesen Gedanken spinnen wir nun weiter: Wir teilen g_1 und g_2 mit Rest durch f und erhalten (eindeutige) Darstellungen $g_1 = q_1 f + r_1$ und $g_2 = q_2 f + r_2$ mit Polynomen $g_1, g_2, r_1, r_2 \in K[t]$ und $\deg r_1, \deg r_2 < \deg f$. Es folgt

$$g_1 - g_2 = (q_1 - q_2)f + (r_1 - r_2) \in (f).$$

   Nun gilt $g_1 - g_2 \in (f)$ genau dann, wenn $g_1 - g_2$ ein Vielfaches von f ist. Dies wiederum ist genau dann der Fall, wenn $r_1 = r_2$ gilt. Das heißt, die beiden Restklassen $g_1 + (f)$ und $g_2 + (f)$ sind genau dann gleich, wenn die Reste von g_1 und g_2 bei Polynomdivision modulo f übereinstimmen.

   Wir stellen uns den Restklassenring modulo f somit als die Menge der Reste bei Polynomdivision modulo f vor: Man kann im Restklassenring $K[t]/(f)$ rechnen wie mit Polynomen, allerdings hat man die genauen Polynome „vergessen“ und kennt nur noch ihren Rest modulo f.

Der folgende Satz, ein Korollar aus der obigen Argumentation, passt zur beschriebenen Anschauung des Restklassenrings.

Satz 2.52 *Sei K ein Körper und $0 \neq f \in K[t]$. Dann gilt*

$$K[t]/(f) = \{r + (f) \mid r \in K[t], \deg r < \deg f\}.$$

Für zwei verschiedene Polynome $r_1, r_2 \in K[t]$ mit $\deg r_1, \deg r_2 < \deg f$ sind auch die Restklassen $r_1 + (f) \neq r_2 + (f)$ verschieden. ◁

Beispiel 2.53 Sei $K = \mathbb{Z}_2$ und $f = t^2 + t + [1] \in \mathbb{Z}_2[t]$. Die Reste bei Polynomdivision modulo f sind $[0], [1], t$ und $t + [1]$. Also gilt

$$Z_2[t]/(f) = \{[0] + (f), [1] + (f), t + (f), t + [1] + (f)\}.$$

Für die Multiplikation gilt

$$(t + [1] + (f)) \cdot (t + (f)) = t^2 + t + (f) = [1] + (f),$$

da der Rest von $t^2 + t$ nach Division mit Rest durch f gleich $[1]$ ist. ◁

Konstruktion 2.54 (Inverse in Restklassenring) Sei K ein Körper und $f \in K[t]$ irreduzibel. Der Restklassenring $K[t]/(f)$ ist dann ein Körper. Sei außerdem $0 \neq g \in K[t] \setminus (f)$ ein beliebiges Polynom. Wir werden nun das Inverse von $0 \neq g + (f) \in K[t]/(f)$ bezüglich der Multiplikation konstruieren.

Da f irreduzibel ist und $g \notin (f)$ gilt, ist $1 \in K$ ein größter gemeinsamer Teiler von f und g. Somit kann man mit dem erweiterten euklidischen Algorithmus, siehe Bemerkung 2.44, das neutrale Element $1 \in K$ der Multiplikation als Linearkombination

$$a_1 \cdot f + a_2 \cdot g = 1 \quad \in K[t]$$

von f und g über $K[t]$ darstellen. Es folgt

$$1 + (f) = a_1 \cdot f + a_2 \cdot g + (f) = a_2 \cdot g + (f) = (a_2 + (f)) \cdot (g + (f)).$$

Somit ist $a_2 + (f)$ das Inverse von $g + (f)$ in $K[t]/(f)$ bezüglich der Multiplikation. ◁

Der nächste Satz beantwortet die Frage, wie viele Elemente der Restklassenring $K[t]/(f)$ hat. Der Satz ist ein Korollar von Satz 2.52

Satz 2.55 *Sei $f \in K[t]$ ein Polynom über dem Körper K mit Grad $n = \deg f$.*

1. *Das Tripel $(K[t]/(f), +, \cdot_K)$ ist ein Vektorraum über K von Dimension n. Dabei ist die Skalarmultiplikation $\cdot_K : K \times K[t]/(f) \to K[t]/(f)$ durch*

$$(a, \sum_{k=0}^{m} a_k t^k) \mapsto \sum_{k=0}^{m} (a \cdot a_k) t^k$$

   definiert. Eine Basis des K-Vektorraums $K[t]/(f)$ ist durch $1, t + (f), \ldots, t^{n-1} + (f)$ gegeben.

2. *Die Abbildung*

$$\psi : K[t]/(f) \to K^n, \quad \sum_{k=0}^{n-1} a_k t^k + (f) \mapsto (a_0, a_1, \ldots, a_{n-1})$$

   ist ein Isomorphismus von K-Vektorräumen. Insbesondere ist für $|K| < \infty$ die Anzahl der Elemente

$$|K[t]/(f)| = |K|^{\deg f}.$$ ◁

Korollar 2.56 *Seien $p \in \mathbb{P}$ und $f \in \mathbb{Z}_p[t]$ irreduzibel. Dann gilt:*

1. *$\mathbb{Z}_p[t]/(f)$ ist ein Körper mit $p^{\deg f}$ Elementen.*

2. *Als Vektorraum über $\mathbb{Z}_p$ ist $\mathbb{Z}_p[t]/(f)$ isomorph zu $\mathbb{Z}_p^{\deg f}$.* ◁

Bemerkung 2.57 (Klassifikation endlicher Körper) Über die Konstruktion von Restklassenringen haben wir gesehen, wie man endliche Körper konstruieren kann, deren Anzahl von Elementen eine Primzahlpotenz ist. Nun stellen sich folgende Fragen:

1. Für welche Primzahlen p und welche $n \in \mathbb{N}$ existiert ein Körper mit p^n Elementen?

   Man kann zeigen, dass für jedes $p \in \mathbb{P}$ und jedes $n \in \mathbb{N}$ ein irreduzibles Polynom $f_n \in \mathbb{Z}_p[t]$ vom Grad $\deg(f_n) = n$ existiert. Wenn man nun den Restklassenkörper $\mathbb{Z}_p[t]/(f_n)$ betrachtet, dann folgt mit Satz 2.55, dass $\mathbb{Z}_p[t]/(f_n)$ genau $p^{\deg(f_n)} = p^n$ Elemente hat.

   Anders formuliert: Für jede Primzahl $p \in \mathbb{N}$ und jedes $n \in \mathbb{N}$ existiert ein Körper mit p^n Elementen.

2. Gibt es endliche Körper, deren Mächtigkeit keine Primzahlpotenz ist?

   Man kann zeigen, dass die Anzahl der Elemente eines endlichen Körpers immer eine Primzahlpotenz ist, d. h. für jeden endlichen Körper K existiert eine Primzahl $p \in \mathbb{N}$ und ein $n \in \mathbb{N}$ mit $|K| = p^n$.

3. Seien $n \in \mathbb{N}$, $p \in \mathbb{P}$. Wie viele Körper mit p^n Elementen gibt es?

   Man kann zeigen, dass zwei endliche Körper K_1 und K_2, die gleich viele Elemente haben, isomorph sind. Bis auf Isomorphie gibt es für jede Primzahl $p \in \mathbb{N}$ und für jedes $n \in \mathbb{N}$ also genau einen Körper mit p^n Elementen.

Zusammenfassend ergibt sich, dass alle endlichen Körper zu einem Restklassenring

$$\mathbb{Z}_p[t]/(f)$$

isomorph sind (für ein $p \in \mathbb{P}$ und ein irreduzibles Polynom $f \in \mathbb{Z}_p[t]$). ◁

Aufgabe 2.58 Es sei $p \in \mathbb{N}$ eine Primzahl. Zeigen Sie, dass ein Polynom $f \in \mathbb{Z}_p[t]$ existiert, für das $\mathbb{Z}_p$ und $\mathbb{Z}_p[t]/(f)$ als Körper isomorph sind. ◁

Notation 2.59 (Endliche Körper) Sei $p \in \mathbb{N}$ eine Primzahl und $n \in \mathbb{N}$. Dann bezeichnen wir mit $\mathrm{GF}(p^n)$ den (laut Bemerkung 2.57) bis auf Isomorphie eindeutigen Körper mit p^n Elementen. ◁

Bemerkung 2.60 (Das Alphabet von AES) Als Alphabet des Secret-Key-Kryptosystems AES wird der Körper $\mathrm{GF}(2^8)$ in der folgenden Darstellung genutzt:

Das Polynom

$$f_{\mathrm{AES}} = t^8 + t^4 + t^3 + t + 1 \in \mathbb{Z}_2[t]$$

ist irreduzibel und vom Grad $\deg(f_{\mathrm{AES}}) = 8$. Somit ist

$$\mathrm{GF}(2^8) = \mathbb{Z}_2[t]/(f_{\mathrm{AES}})$$

ein Körper mit $2^8 = 256$ Elementen. Ein Element $f = \sum_{i=0}^{7} a_i t^i \in \mathbb{Z}_2[t]/(f_{\mathrm{AES}})$ wurde in der Beschreibung von AES in Abschnitt 2.2 mit

$$\Psi(f) = \Psi\Big(\sum_{i=0}^{7} a_i t^i\Big) = (a_0, \ldots, a_7) \in \mathbb{Z}_2^8$$

identifiziert, wobei Ψ die Abbildung aus 2.55 bezeichnet. ◁

3. Public-Key-Kryptosysteme

Bei der Verwendung von Secret-Key-Kryptosystemen zur Verschlüsselung von Botschaften kennen sowohl Sender als auch Empfänger den geheimen Verschlüsselungsschlüssel $e \in \mathscr{K}$, mit dem ein Klartext $p \in \mathscr{P}$ zu einem Chiffretext $c = E_e(p)$ verschlüsselt wird. Dieser geheime Schlüssel e muss vor seiner Verwendung bei einem Treffen zwischen Sender und Empfänger ausgetauscht werden. Aus dem Verschlüsselungsschlüssel e können beide „leicht“ einen zugehörigen Entschlüsselungsschlüssel $d(e) \in \mathscr{K}$ berechnen und den Chiffretext c zu $D_{d(e)}(c) = p$ entschlüsseln. Verschlüsselung mittels Secret-Key-Kryptosystemen hat somit den Nachteil, dass der geheime Schlüssel e irgendwann vor der Kommunikation ausgetauscht worden sein muss; anders ausgedrückt: Es muss ein Treffen zwischen Sender und Empfänger stattgefunden haben.

In Public-Key-Kryptosystemen ist das anders. Hier können verschlüsselte Nachrichten übermittelt werden, ohne dass vorher ein Austausch eines geheimen Schlüssels zwischen Sender und Empfänger stattgefunden hat. Der Austausch eines geheimen Schlüssels wird überflüssig, da bei Public-Key-Kryptosystemen der Verschlüsselungsschlüssel $e \in \mathscr{K}$ öffentlich, das heißt frei zugänglich ist – daher der Name „Public-Key“. Da jeder Zugriff auf den Verschlüsselungsschlüssel $e \in \mathscr{K}$ hat, kann jeder beliebige Botschaften $p \in \mathscr{P}$ zu Chiffretexten $c = E_e(p)$ verschlüsseln.

Dieses Wissen um den Verschlüsselungsschlüssel stärkt die Position eines Angreifers: Ein Angreifer fängt den Chiffretext $c = E_e(p) \in \mathscr{C}$ ab, kennt den genutzten Verschlüsselungsschlüssel e und kann beliebige Klartexte $p \in \mathscr{P}$ verschlüsseln. Bei Public-Key-Kryptosystemen stehen einem Angreifer somit alle Zutaten für einen Chosen-Plaintext-Angriff zur Verfügung.

Diesen „Nachteil“ machen Public-Key-Kryptosysteme durch eine andere Eigenschaft wett: In Public-Key-Systemen ist es „unzulässig“ aus der Kenntnis des Verschlüsselungsschlüssels e den zugehörigen Entschlüsselungsschlüssel $d(e)$ zu berechnen, d. h.: Die Berechnung von $d(e)$ aus e würde in aller Regel so lange dauern, dass die Berechnung von $d(e)$ in der Praxis unmöglich ist.

Da die Algorithmen von Secret-Key-Systemen in der Regel effizienter sind als die Algorithmen von Public-Key-Systemen, wird in der Praxis meist ein Public-Key-System genutzt, um einen geheimen Schlüssel auszutauschen. Die anschließende geheime Kommunikation erfolgt dann über ein Secret-Key-System mithilfe des bereits ausgetauschten Schlüssels.

Laufzeiten von Algorithmen

Definition 3.1 ($\mathcal{O}$-Notation) Es sei $k \in \mathbb{N}$ und $f : \mathbb{N}^k \to \mathbb{R}$, $g : \mathbb{N}^k \to \mathbb{R}$ seien Funktionen. Man schreibt $f = \mathcal{O}(g)$, wenn positive Zahlen $C, z_1, \dots, z_k \in \mathbb{N}$ existieren, so dass für alle $(n_1, \dots, n_k) \in \mathbb{N}^k$ mit $n_i > z_i$, $1 \le i \le k$, Folgendes, gilt:

$$f(n_1, \dots, n_k) \le C \cdot g(n_1, \dots, n_k)$$

Anschaulich heißt das, dass „fast überall" $f(n_1, \dots, n_k) \le C \cdot g(n_1, \dots, n_k)$ erfüllt ist. Ist g eine konstante Funktion, so schreibt man $f = \mathcal{O}(1)$ (da dann $f(n_1, \dots, n_k) \le C \cdot 1$ gilt). ◁

Beispiel 3.2

- $2n^2 + n + 1 = \mathcal{O}(n^2)$, da $2n^2 + n + 1 \le 4 \cdot n^2$ für alle $n \ge 1$.
- $3nm + n + m + 1 = \mathcal{O}(nm)$, da $3mn + n + m + 1 \le 6 \cdot mn$ für alle $n, m \ge 1$. ◁

Definition 3.3 (Polynomieller Algorithmus)

1. Sei $n \in \mathbb{N}$ eine Zahl. Dann bezeichnen wir mit $\text{size}(n)$ die Anzahl der Ziffern in ihrer Binärdarstellung. (Man kann $\text{size}(n)$ auch als $\lfloor \log_2(n) \rfloor + 1$, den abgerundeten Logarithmus von n zur Basis 2 plus 1, beschreiben.)
2. Sei A ein Algorithmus, der natürliche Zahlen $z_1, \dots, z_n \in \mathbb{N}$ als Input hat. Wir nennen A *polynomiell*, wenn $e_1, \dots, e_n \in \mathbb{N}$ existieren, so dass A die Laufzeit

   $$\mathcal{O}\left(\text{size}(z_1)^{e_1} \cdot \dots \cdot \text{size}(z_n)^{e_n}\right)$$

   hat.
3. Man sagt, dass ein Problem in der Komplexitätsklasse P liegt, wenn es durch einen polynomiellen Algorithmus gelöst werden kann. ◁

Beispiel 3.4 (Addition ganzer Zahlen) Die Addition ganzer Zahlen $n, m \in \mathbb{Z}$ ist in Polynomialzeit möglich, hier ein Beispiel:

Seien $n = 49$, $m = 23$ mit Binärdarstellung $b_n = 110001$ und $b_m = 10111$. Dann ist $\text{size}(n) = 6$ und $\text{size}(m) = 5$. Um m und n zu addieren, müssen $6 = \max\{\text{size}(m), \text{size}(n)\}$ Bit-Additionen mit Übertrag durchgeführt werden:

		1	1	0	0	0	1
+			1	0	1	1	1
Übertrag	1	1		1	1	1	
	1	0	0	1	0	0	0

Wir nehmen an, dass die Addition zweier Bits Zeit $\mathcal{O}(1)$, also konstante Zeit braucht. Daher braucht die gesamte Addition von n und m Zeit $\mathcal{O}(\max\{\text{size}(n), \text{size}(m)\})$. ◁

Beispiel 3.5 Folgende Problemstellungen können mithilfe eines polynomiellen Algorithmus gelöst werden:

- Addition ganzer Zahlen
- Multiplikation ganzer Zahlen
- Division mit Rest
- Erweiterter Euklidischer Algorithmus
- Berechnung von Inversen in $\mathbb{Z}_n$ ◁

Probleme, die mithilfe eines polynomiellen Algorithmus gelöst werden können, werden als *praktisch lösbar* betrachtet. Das liegt an folgendem Sachverhalt: Wenn die Eingabegröße eines Algorithmus (sprich: die Zahlen, die er als Eingabe erhält) wächst, steigt in der Regel auch die Laufzeit des Algorithmus. Für nicht-polynomielle Algorithmen steigt die Laufzeit deutlich schneller an, als die Eingabegröße wächst. Dadurch kann es passieren, dass ein Problem bereits nach einer geringen Vergrößerung der Eingabegröße nicht mehr in vertretbarer Zeit gelöst werden kann. Bei polynomiellen Algorithmen passiert so etwas (bei kleinen Exponenten e_i und einer kleinen Konstante C) nicht. Liegt kein polynomieller Lösungsalgorithmus vor, wird ein Problem als *praktisch unlösbar* betrachtet, auch wenn man theoretisch weiß, wie es gelöst werden könnte.

Definition: Public-Key-Kryptosysteme

Mithilfe des Begriffs eines polynomiellen Algorithmus verfeinern wir nun die Definition eines Public-Key-Kryptosystems.

Definition 3.6 (Public-Key-Kryptosystem) Ein *Public-Key-Kryptosystem* ist ein Kryptosystem, in dem der Schlüsselraum $\mathcal{K} = \mathcal{K}_E \dot{\cup} \mathcal{K}_D$ aus zwei Teilmengen besteht, dem Verschlüsselungsraum $\mathcal{K}_E$ und dem Entschlüsselungsraum $\mathcal{K}_D$, so dass folgende Eigenschaften erfüllt sind:

1. Für jeden Verschlüsselungsschlüssel $e \in \mathcal{K}_E$ existiert ein Entschlüsselungsschlüssel $d(e) \in \mathcal{K}_D$, so dass $D_{d(e)}(E_e(p)) = p$ für alle Klartexte $p \in \mathcal{P}$ gilt.
2. Zur Auswertung aller Verschlüsselungsfunktionen $E_e \in \mathcal{E}$ und Entschlüsselungsfunktionen $D_d \in \mathcal{D}$ existieren Algorithmen mit polynomieller Laufzeit.
3. Der genutzte Verschlüsselungsschlüssel $e \in \mathcal{K}_E$ wird veröffentlicht.
4. Es existiert kein polynomieller Algorithmus, der aus einem Verschlüsselungsschlüssel $e \in \mathcal{K}_E$ und aus der Kenntnis des Kryptosystems den zugehörigen Entschlüsselungsschlüssel $d(e) \in \mathcal{K}_D$ berechnen kann. ◁

Bemerkung 3.7 In einem Public-Key-Kryptosystem existiert kein polynomieller Algorithmus, mit dem allein durch die Kenntnis eines Verschlüsselungsschlüssels $e \in \mathcal{K}_E$ der zugehörige Entschlüsselungsschlüssel $d(e) \in \mathcal{K}_D$ berechnet werden kann. Es ist also „praktisch unmöglich“ $d(e)$ aus e zu berechnen. Ohne die Kenntnis des Entschlüsselungsschlüssels $d(e)$ kann (in der Regel) auch ein Chiffretext $c = E_e(p) \in \mathcal{C}$ nicht in Polynomialzeit zu $p = D_{d(e)}(c) = D_{d(e)}(E_e(p)) \in \mathcal{P}$ entschlüsselt werden.

Wenn der Entschlüsselungsschlüssel $d(e)$ bekannt ist, ändert sich das: Ist $d(e)$ bekannt, kann $p = D_{d(e)}(c)$ laut Definition eines Public-Key-Kryptosystems in Polynomialzeit berechnet werden. Man sagt, dass $d(e)$ *Trapdoor-Information* enthält (engl. Falltür). Durch die Trapdoor-Information wird die Invertierung der Verschlüsselungsfunktion E_e zulässig. ◁

Konstruktion 3.8 (Kommunikation mittels Public-Key-Kryptosystemen) Bob möchte geheime Nachrichten empfangen können, ohne vorher mit dem Sender der Nachricht einen geheimen Schlüssel austauschen zu müssen. Dies ist mithilfe eines Public-Key-Kryptosystems möglich:

Bob wählt einen öffentlichen Schlüssel $e \in \mathcal{K}_E$ und einen zugehörigen geheimen Schlüssel $d(e) \in \mathcal{K}_D$. Dann veröffentlicht er den geheimen Schlüssel e. Nun kann Alice, die Bob bisher nicht kennt und nie getroffen hat, eine beliebige Nachricht $p \in \mathcal{P}$ zu

$$c = E_e(p) \in \mathcal{C}$$

verschlüsseln und an Bob senden. Da Bob seinen persönlichen geheimen Schlüssel $d(e)$ kennt, kann er Alices Nachricht c zu

$$p = D_{d(e)}(c) = D_{d(e)}(E_e(p))$$

entschlüsseln. Ver- und Entschlüsselung erfolgen jeweils in Polynomialzeit.

Die Sicherheit des Kryptosystems beruht darauf, dass die Berechnung von $d(e)$ aus e nicht in Polynomialzeit möglich und damit „praktisch unmöglich“ ist: Eine Angreiferin Eve braucht unvertretbar lange um $d(e)$ aus e zu berechnen und somit (in der Regel) auch unvertretbar lange um c zu $p = E_{d(e)}(c)$ zu entschlüsseln. ◁

Bemerkung 3.9 Es gibt Kryptosysteme, bei denen die Ver- oder Entschlüsselungsalgorithmen keine Funktionen, sondern sogenannte Multifunktionen sind. Ein Beispiel ist das Rabin-Kryptosystem, das im Rahmen dieser Vorlesung aus Zeitgründen jedoch nicht vorgestellt wird. Eine Multifunktion $f : M \multimap N$ ist eine Abbildung $\tilde{f} : M \to \mathrm{Pot}(N)$, d. h. ein Element $m \in M$ des Definitionsbereichs wird auf eine Teilmenge des Wertebereichs abgebildet.

Ist eine Entschlüsselungsfunkion $D_d : \mathcal{C} \multimap \mathcal{P}$ eine Multifunktion, so wird ein Chiffretext $c \in \mathcal{C}$ zu einer Menge von Klartexten $D_d(c) = \{p_1, \dots, p_n\} \subset \mathcal{P}$ entschlüsselt – und nicht nur zu einem. Um c zu entschlüsseln, muss aus dieser Menge von Klartexten $\{p_1, \dots, p_n\}$ händisch der „richtige“ ausgewählt werden, d. h. man muss prüfen, welcher der (hoffentlich eindeutige) Klartext p_i ist, der Sinn ergibt. ◁

Kandidaten für Einwegfunktionen

Public-Key-Kryptosysteme bauen auf sogenannten Einwegfunktionen auf, die wir nun einführen.

Definition 3.10 Seien $k, l \in \mathbb{N}$. Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}^l$ heißt *Einwegfunktion*, wenn $f(x)$ für alle $x \in \mathbb{N}^k$ in Polynomialzeit berechnet werden kann und wenn gleichzeitig kein polynomieller Algorithmus existiert, der Urbilder unter f berechnet. ◁

Anschaulich gesprochen erfüllt eine Einwegfunktion $f : \mathbb{N} \to \mathbb{N}$ folgende Eigenschaft: Das Berechnen von Bildern $f(x)$ ist „einfach“, also in Polynomialzeit möglich; das Berechnen von Urbildern jedoch ist „schwierig“, also nicht in Polynomialzeit möglich, und dauert „unvertretbar lange“.

Bemerkung 3.11 Man kann sich das Nachschlagen einer Telefonnummer in einem klassischen Papiertelefonbuch als Einwegfunktion vorstellen. Kennt man den Namen einer Person, kann man dank der alphabetischen Sortierung recht schnell und einfach herausfinden, was ihre Telefonnummer ist. Wenn man allerdings herausfinden möchte, welche Person hinter einer gegebenen Nummer steckt, muss man das gesamte Telefonbuch durchgehen, was unvertretbar lange dauert. ◁

Bemerkung 3.12 Bis heute ist nicht bekannt, ob Einwegfunktionen überhaupt existieren! Auch wenn es vielversprechende Kandidaten für solche Einwegfunktionen gibt, konnte ihre Existenz bis heute nicht bewiesen werden. Damit ist ebenfalls unklar, ob Public-Key-Kryptosysteme existieren: Nur unter Annahmen kann gezeigt werden, dass ein Kryptosystem die Eigenschaft erfüllt, dass der zu $e \in \mathcal{K}_E$ gehörige Entschlüsselungsschlüssel $d(e)$ nicht in Polynomialzeit bestimmt werden kann.

Man kann zeigen, dass aus der Existenz von Einwegfunktionen folgen würde, dass die Komplexitätsklassen P und NP verschieden sind. P ist die Klasse der Probleme, die in polynomieller Zeit lösbar sind; NP die Klasse der Probleme, für die in polynomieller Zeit überprüft werden kann, ob ein vorgegebenes Ergebnis das Problem tatsächlich löst.

Zwei Beispiele:

Das Problem der Addition zweier Zahlen $n, m \in \mathbb{N}$ ist, wie wir gesehen haben, in polynomieller Zeit lösbar, d. h. das Ergebnis $n + m \in \mathbb{N}$ kann in polynomieller Zeit bestimmt werden. Dieses Problem liegt also in der Komplexitätsklasse P. Außerdem kann in polynomieller Zeit überprüft werden, ob eine Zahl $a \in \mathbb{N}$ die Summe von n und m ist, z. B. indem man erst $n + m$ in polynomieller Zeit berechnet und dann überprüft, ob $a = n + m$ gilt. Damit liegt das Problem der Addition zweier Zahlen $n, m \in \mathbb{N}$ auch in NP.

Nun betrachten wir das Problem, bei dem in einem Graphen ein geschlossener Weg gesucht wird, der jeden Knoten genau einmal durchläuft. Ein solcher Weg heißt Hamiltonkreis. Die Fragestellung, ob ein Hamiltonkreis in einem Graphen existiert, wird Hamiltonkreisproblem genannt. Es liegt in NP: Wenn ein Weg im Graphen gegeben ist, kann durch Durchlaufen des

gesamten Weges getestet werden, ob jeder Knoten genau einmal durchlaufen wird und ob der Weg geschlossen ist. Ein Durchlaufen des Weges ist in Polynomialzeit möglich, somit kann in Polynomialzeit getestet werden, ob ein Weg ein Hamiltonkreis ist. Ob das Problem auch in P liegt ist unbekannt: Bisher ist die Frage ungelöst, ob ein Algorithmus existiert, der die Frage „Existiert in einem Graphen ein Hamiltonkreis?“ in Polynomialzeit beantwortet.

Der Buchstabe P steht, wie zu erwarten, für „polynomial time“: Eine Lösung kann in Polynomialzeit gefunden werden. Die Buchstaben NP stehen für „non-deterministic polynomial time“: Von einem vorgegebenen Ergebnis kann in Polynomialzeit überprüft werden, ob es tatsächlich eine Lösung ist.

Ob P = NP gilt, ist bis heute unklar. Diese Fragestellung ist eins der sieben Millenium-Probleme, für deren Lösung das Clay Mathematics Institute im Jahr 2000 jeweils eine Million US-Dollar ausgelobt hat. Bisher konnte erst eins der Probleme gelöst werden, durch den Beweis der sogenannten Poincaré-Vermutung im Jahr 2002 durch den russischen Mathematiker Grigori Jakowlewitsch Perelman. ◁

Konstruktion 3.13 (Kandidaten für Einwegfunktionen)

1. *Das Faktorisierungsproblem (FP)*: Sei $n \in \mathbb{N}$. Finde eine Primzahl $p \in \mathbb{N}$, die n teilt.
2. *Das diskrete Logarithmus-Problem (DLP)*: Sei $G = \langle g \rangle = \{g^0, g^1, g^2, \dots\}$ eine endliche zyklische Gruppe der Ordnung $n \in \mathbb{N}$ und $x \in G$. Finde ein $a \in \mathbb{N}$, so dass $g^a = x$ gilt. Eine solche Zahl $a \in \mathbb{N}$ nennt man *diskreten Logarithmus von x zur Basis $g \in G$*. ◁

Die beiden beschriebenen Probleme liegen in der Komplexitätsklasse NP: Von einer Zahl $a \in \mathbb{N}$ kann in Polynomialzeit überprüft werden, ob sie ein Teiler von $n \in \mathbb{N}$ ist; eben indem man eine Division von n durch a mit Rest durchführt und testet, ob der Rest gleich Null ist. Ebenso kann in Polynomialzeit überprüft werden, ob eine Zahl $a \in \mathbb{N}$ und ein Gruppenelement $x \in \langle g \rangle$ die Eigenschaft $g^a = x$ erfüllen; eben indem man g^a berechnet und testet, ob $g^a = x$ gilt.

Von keinem der beiden Probleme ist bekannt, ob sie auch in der Komplexitätsklasse P liegen: Es ist unbekannt, ob ein polynomieller Algorithmus existiert, mit dem ein Teiler $a \in \mathbb{N}$ von $n \in \mathbb{N}$ gefunden werden kann. Ebenso ist es unbekannt, ob ein polynomieller Algorithmus existiert, mit dem für eine endliche Gruppe $G = \langle g \rangle$ ein diskreter Logarithmus von $x \in G$ zur Basis g berechnet werden kann.

Man trifft in der Kryptographie jedoch folgende Standardannahmen. Nur unter solchen Annahmen kann die Existenz kryptographisch wertvoller Einwegfunktionen und die Existenz von Public-Key-Kryptosystemen bewiesen werden.

Annahme 3.14

- *Faktorisierungs-Annahme (FA)*: Das Faktorisierungsproblem liegt **nicht** in der Komplexitätsklasse P.
- *Logarithmus-Annahme (LA)*: Das diskrete Logarithmus-Problem liegt **nicht** in der Komplexitätsklasse P. ◁

3.1. Das RSA-Verfahren

Das RSA-Verfahren ist nach seinen Erfindern Ron Rivest, Adi Shamir und Len Adleman benannt. Es war das erste Public-Key-Kryptosystem und wurde 1983 als Patent angemeldet. Noch heute ist es das wichtigste Public-Key-Kryptosystem. Seine Sicherheit beruht u. a. auf der Faktorisierungs-Annahme.

Die folgende Funktion sollte aus der Geometrie-Vorlesung bekannt sein:

Notation 3.15 Wir definieren die Eulersche φ-Funktion $\varphi : \mathbb{N} \to \mathbb{N}$ durch

$$n \mapsto |\mathbb{Z}_n^\times|,$$

wobei $\mathbb{Z}_n^\times$ für die Einheitengruppe von $\mathbb{Z}_n$ steht, d. h. $\mathbb{Z}_n^\times$ enthält alle $[a] \in \mathbb{Z}_n$, für die $\mathrm{ggT}(a, n) = 1$ gilt. Die Zahl $\varphi(n)$ gibt für $n \in \mathbb{N}$ die Anzahl der natürlichen Zahlen an, die kleiner als n und teilerfremd zu n sind. ◁

Aufgabe 3.16 Seien $q_1, q_2 \in \mathbb{N}$ zwei verschiedene Primzahlen. Dann gilt $\varphi(q_1 q_2) = (q_1 - 1)(q_2 - 1)$. ◁

Definition 3.17 (RSA-Problem) *Das RSA-Problem (RSAP)*: Seien $n, p, e \in \mathbb{N}$ mit

$$\mathrm{ggT}(e, \varphi(n)) = 1.$$

Gegeben n, e und $p^e \mod n$, bestimme $p \mod n$. ◁

Annahme 3.18 (RSA-Annahme) Wir treffen die anerkannte *RSA-Annahme (RSAA)*: Das RSA-Problem liegt nicht in der Komplexitätsklasse P. ◁

Definition 3.19 (RSA-Kryptosystem) Das *RSA-Kryptosystem* ist wie folgt definiert:

1. Klartext- und Chiffretextraum sind $\mathcal{P} = \{0, \dots, 2^k - 1\}, \mathcal{C} = \{0, \dots, 2^{k+1} - 1\}$ für ein $k \in \mathbb{N}$ (häufig $k = 1024$).
2. Es gilt $\mathcal{K}_E = \{(n = q_1 q_2, e) \mid \mathrm{size}(n) = k + 1,\ q_1 \neq q_2 \text{ prim},\ e \in \mathbb{N},\ \mathrm{ggT}(e, \varphi(n)) = 1\}$.
3. Es gilt $\mathcal{K}_D = \{(n, d, e) \mid (n, e) \in \mathcal{K}_E, 0 < d \leq \varphi(n), de \equiv 1 \mod \varphi(n)\}$.
4. Die zu $(n, e) \in \mathcal{K}_E$ gehörige Verschlüsselungsfunktion ist
$$E_{(n,e)} : \mathcal{P} \to \mathcal{C},\ \ p \mapsto p^e \mod n.$$
5. Die zu $(n, d, e) \in \mathcal{K}_D$ gehörige Entschlüsselungsfunktion ist
$$D_{(n,d,e)} : \mathcal{C} \to \mathcal{P},\ \ c \mapsto c^d \mod n.$$ ◁

Beispiel 3.20 Es seien $q_1 = 11, q_2 = 23$ und $e = 3$. Dann ist

- $n = q_1 q_2 = 253$ mit Binärdarstellung 11111101,
- $k + 1 = \text{size}(n) = 8$,
- $\varphi(n) = (p-1)(q-1) = 10 \cdot 22 = 220$ und
- $d = 147$ mit $e \cdot d = 441 \equiv 1 \mod 220$.

Für den Chiffretext $p = 52$ erhalten wir $E_{(253,3)}(p) \equiv 52^3 \equiv 193 \mod 253$. Um c zu entschlüsseln, berechnen wir

$$D_{(253,147,3)}(c) \equiv c^d \equiv c^{147} \equiv 193^{147} \equiv 52 \mod 253. \qquad \triangleleft$$

Satz 3.21 *Unter der RSA-Annahme ist das RSA-Kryptosystem ein Public-Key-Kryptosystem.* $\triangleleft$

Um diesen Satz zu beweisen, benötigen wir einige zahlentheoretische Resultate, u. a. den folgenden Satz, der bereits in der Geometrie-Vorlesung bewiesen wurde:

Satz 3.22 (Kleiner Satz von Fermat) *Sei G eine (multiplikativ geschriebene) endliche Gruppe mit neutralem Element e_G. Dann gilt $a^{|G|} = e_G$ für alle $a \in G$.* $\triangleleft$

Lemma 3.23 *Seien $q_1, q_2 \in \mathbb{P}$ und $n = q_1 q_2$. Seien außerdem $d, e \in \mathbb{N}$ mit $de \equiv 1 \mod \varphi(n)$. Dann gilt:*

1. *Die Abbildung*
$$\Phi : \mathbb{Z}_n \to \mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}, \quad [a]_n \mapsto ([a]_{q_1}, [a]_{q_2})$$
ist ein Ringisomorphismus.
2. $p^{d \cdot e} \equiv p \mod n$ *für alle $p < n$.* $\triangleleft$

Beweis. 1. Φ ist ein Ringhomomorphismus, denn für alle $[a]_n, [b]_n \in \mathbb{Z}_n$ gilt

$$\Phi([a]_n + [b]_n) = ([a+b]_{q_1}, [a+b]_{q_2}) = ([a]_{q_1}, [a]_{q_2}) + ([b]_{q_1}, [b]_{q_2}) = \Phi([a]_n) + \Phi([b]_n)$$

und

$$\Phi([a]_n \cdot [b]_n) = ([a \cdot b]_{q_1}, [a \cdot b]_{q_2}) = ([a]_{q_1}, [a]_{q_2}) \cdot ([b]_{q_1}, [b]_{q_2}) = \Phi([a]_n) \cdot \Phi([b]_n).$$

Außerdem ist Φ injektiv, denn für alle $[a]_n, [b]_n \in \mathbb{Z}_n$ gilt:

$\Phi([a]_n) = \Phi([b]_n) \Rightarrow ([a]_{q_1}, [a]_{q_2}) = ([b]_{q_1}, [b]_{q_2}) \Rightarrow ([a-b]_{q_1}, [a-b]_{q_2}) = ([0]_{q_1}, [0]_{q_2}) \Rightarrow q_1, q_2 | a - b \Rightarrow n = q_1 q_2 | a - b \Rightarrow [a-b]_n = [0]_n \Rightarrow [a]_n = [b]_n$.

Da $|\mathbb{Z}_n| = n = q_1 q_2 = |\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}| < \infty$ gilt, folgt aus der Injektivität von Φ, dass Φ sogar bijektiv ist.

2. Wegen $de \equiv 1 \mod \varphi(n)$ existiert $\lambda \in \mathbb{Z}$ mit $de = 1 + \lambda \cdot \varphi(n)$. Mit Aufgabe 3.16 folgt

$$p^{d\cdot e} = p^{1+\lambda\varphi(n)} = p \cdot (p^{(q_1-1)(q_2-1)})^{\lambda}$$

und mit dem kleinen Satz von Fermat

$$p^{d\cdot e} \equiv p \mod q_1 \quad \text{und} \quad p^{d\cdot e} \equiv p \mod q_2. \tag{3.1}$$

Aus Teil a) wissen wir, dass Φ ein Ringisomorphismus ist. Somit existiert Φ^{-1} und wir schließen

$$([p]_n)^{de} = [p^{de}]_n = \Phi^{-1}([p^{de}]_{q_1}, [p^{de}]_{q_2}) = \Phi^{-1}([p]_{q_1}, [p]_{q_2}) = [p]_n. \qquad \square$$

Beweis (Satz 3.21***).*** Wir prüfen die in der Definition eines Public-Key-Kryptosystems 3.6 gelisteten Eigenschaften.

1. Seien $(n, e) \in \mathcal{K}_E$ und $(n, d, e) \in \mathcal{K}_D$ mit $d \cdot e \equiv 1 \mod \varphi(n)$. Für alle $p \in \mathcal{P}$ ergibt sich mit Lemma 3.23

$$D_{(n,d,e)}(E_{(n,e)}(p)) \equiv D_{(n,d,e)}(p^e \mod n) \equiv (p^e)^d \equiv p \mod n.$$

Somit ist $D_{(n,d,e)}$ die zu $E_{(n,d)}$ gehörige Entschlüsselungsfunktion.

2. Es existieren polynomielle Algorithmen, um die Ver- und Entschlüsselungsfunktionen $D_{(n,d,e)}$ und $E_{(n,e)}$ auszuwerten: Das Berechnen von Potenzen $p^e \mod n$ entspricht

$$\mathcal{O}(\text{size}(e))$$

Multiplikationen modulo n und ist somit in Polynomialzeit möglich (zur Erinnerung: Multiplikation und Division mit Rest modulo n ist in Polynomialzeit möglich).

3. Wir zeigen, dass unter der RSA-Annahme kein polynomieller Algorithmus existiert, mit dem der zu $(n, e) \in \mathcal{K}_E$ gehörige Entschlüsselungsschlüssel $(n, d, e) \in \mathcal{K}_D$ in Polynomialzeit bestimmt werden kann:

Gäbe es einen Algorithmus, der (n, d, e) aus (n, e) in Polynomialzeit berechnet, dann könnte $E_e(p) = p^e \in \mathcal{C}$ für alle $p \in \mathcal{P}$ in Polynomialzeit zu

$$D_{(n,d,e)}(p^e) = (p^e)^d \equiv p \mod n$$

entschlüsselt werden (denn $D_{(n,d,e)}$ kann laut 2. in Polynomialzeit ausgewertet werden). Gegeben $n, e \in \mathbb{N}$ und $p^e \mod n$, könnte also p in Polynomialzeit berechnet werden. Dies widerspricht der RSA-Annahme. $\square$

Wir analysieren nun die Sicherheitseigenschaften des RSA-Verfahrens unter einem Chosen-Plaintext-Angriff, siehe dazu 1.12. Wir bezeichnen den genutzten öffentlichen Schlüssel mit $n = (q_1, q_2, e) \in \mathcal{K}_E$.

1. Um die Sicherheit des RSA-Verfahrens zu gewährleisten, müssen die Primzahlen q_1, q_2 zufällig und gleichverteilt gewählt werden. Außerdem müssen q_1, q_2 und $\varphi(q_1 q_2)$ geheim bleiben:

   Ist q_1 bekannt, kann q_2 mittels Division mit Rest von $n = q_1 q_2$ durch q_1 in Polynomialzeit bestimmt werden. Dann kann auch $\varphi(q_1 q_2) = (q_1 - 1)(q_2 - 1)$ in Polynomialzeit berechnet werden, siehe Aufgabe 3.16. Ist nun $\varphi(q_1 q_2) = \varphi(n)$ bekannt, kann die Zahl $0 < d < \varphi(n)$ mit der Eigenschaft $de \equiv 1 \mod \varphi(n)$ in Polynomialzeit berechnet (euklidischer Algorithmus) und $E_{(n,e)}(p) = p^e$ in Polynomialzeit zu $D_{(n,d,e)}(p^e \mod n) \equiv p^{de} \equiv p \mod n$ entschlüsselt werden.

2. Die Primzahlen $q_1, q_2 \in \mathbb{N}$ sollten so gewählt werden, dass $\text{size}(q_1), \text{size}(q_2) \sim \frac{k}{2}$ gilt. Sind sie zu klein oder zu groß gewählt, können sie schnell durch Ausprobieren als Teiler von $n = q_1 \cdot q_2$ entlarvt werden.

3. Die RSA-Annahme ist stärker als die Faktorisierungs-Annahme, d. h. aus der RSA-Annahme folgt die Faktorisierungs-Annahme:

   Der Verschlüsselungsschlüssel $(n, e) \in \mathcal{K}_E$ und der Chiffretext $p^e \mod n$ seien gegeben. Wir nehmen an, dass die Faktorisierungs-Annahme nicht erfüllt ist. Dann kann die Zahl $n = q_1 q_2 \in \mathbb{N}$ in polynomieller Zeit in ihre Primfaktoren q_1, q_2 zerlegt werden, und man kann $\varphi(n) = (q_1 - 1)(q_2 - 1)$ in polynomieller Zeit berechnen. Für $e \in \mathbb{N}$ mit $\text{ggT}(e, \varphi(n)) = 1$ kann dann mit dem erweiterten euklidischen Algorithmus in polynomieller Zeit ein $d \in \mathbb{N}$ bestimmt werden, das $d \cdot e \equiv 1 \mod \varphi(n)$ erfüllt. Gegeben n, e und $p^e \mod n$ kann somit

   $$D_{(n,d,e)}(p^e \mod n) \equiv p^{d \cdot e} \equiv p \mod n$$

   in Polynomialzeit bestimmt werden. Das ist ein Widerspruch zur RSA-Annahme.

4. Die Sicherheitseigenschaften Nicht-Unterscheidbarkeit und Nicht-Modifizierbarkeit sind nicht erfüllt:

   Das RSA-Verfahren ist deterministisch (d. h. die Ausgabe hängt nur von der Eingabe ab) und der Verschlüsselungsschlüssel (n, e) ist öffentlich. Gegeben ein Chiffretext c und eine Menge von Klartexten $\{p_1, \ldots, p_m\} \in \mathcal{P}$, kann ein Angreifer alle zugehörigen Chiffretexte berechnen. Dadurch findet er heraus, welcher der Klartexte $p_1, \ldots, p_m$ zu c verschlüsselt wurde. Demnach ist die Sicherheitseigenschaft Nicht-Unterscheidbarkeit verletzt.

   Außerdem ist die RSA-Funktion multiplikativ, d. h.

   $$E_{(n,e)}(ab) \equiv (ab)^e \equiv a^e \cdot b^e \equiv E_{(n,e)}(a) \cdot E_{(n,e)}(b) \mod n.$$

   Dadurch kann der Chiffretext so modifiziert werden, dass er dem ursprünglichen Klartext ähnelt. Wurde $p \in \mathcal{P}$ zu $c \in \mathcal{C}$ verschlüsselt, so weiß beispielsweise ein Angreifer (der c abhört hat, aber p nicht kennt!), dass $2p$ zu

   $$E_{(n,e)}(2p) = E_{(n,e)}(2) \cdot E_{(n,e)}(p) = E_{(n,e)}(2) \cdot c$$

   verschlüsselt wird. Dies verletzt die Sicherheitseigenschaft Nicht-Modifizierbarkeit.

3.2. Das ElGamal-Kryptosystem

Neben dem RSA-Verfahren gibt es eine Reihe weiterer Public-Key-Kryptosysteme. In dieser Vorlesung stellen wir nur ein weiteres vor, das ElGamal-Kryptosystem. Das ElGamal-System ist ein *probabilistisches Kryptosystem*: Wenn ein Klartext $p \in \mathcal{P}$ mit $e \in \mathcal{K}_E$ verschlüsselt wird, dann hängt der Chiffretext nicht nur vom Klartext p ab. Bei der Berechnung von $c = E_e(p)$ werden zufällige Wahlen getroffen. Verschlüsselt man $p \in \mathcal{P}$ zweimal hintereinander mit demselben Schlüssel $e \in \mathcal{K}_E$, kann (bzw. wird höchstwahrscheinlich) beim zweiten Mal ein anderer Chiffretext herauskommen als beim ersten Mal.

Wir geben folgenden Satz ohne Beweis an:

Satz 3.24 *Sei $q \in \mathbb{P}$ eine Primzahl. Dann ist die Einheitengruppe $\mathbb{Z}_q^\times$ zyklisch, d. h. es existiert ein Element $g \in \mathbb{Z}_q^\times$, das die Gruppe $\mathbb{Z}_q^\times$ erzeugt, also*

$$\langle g \rangle = \{g^0, g^1, g^2, \dots\} = \mathbb{Z}_q^\times$$

erfüllt. ◁

Definition 3.25 (Diffie-Hellman-Problem) Das *Diffie-Hellman-Problem (DHP)*: Seien $q \in \mathbb{P}$ prim, $g \in \mathbb{Z}_q^\times$ ein Erzeuger (d. h. es gilt $\langle g \rangle = \mathbb{Z}_q^\times$) und $a, b \in \mathbb{Z}$. Gegeben g^a und g^b, berechne g^{ab}. ◁

Annahme 3.26 (Diffie-Hellman-Annahme) Das Diffie-Hellman-Problem liegt **nicht** in der Komplexitätsklasse P. ◁

Bemerkung 3.27 Man kann zeigen, dass die Diffie-Hellman-Annahme stärker als die Logarithmusannahme ist, d. h. wenn die Diffie-Hellman-Annahme gilt, dann ist auch die Logarithmusannahme erfüllt. ◁

Definition 3.28 (ElGamal-Kryptosystem) Das probabilistische *ElGamal-Kryptosystem* ist wie folgt definiert:

1. Klartext- und Chiffretextraum sind

$$\mathcal{P} = \{0, \dots, 2^k - 1\} \text{ und } \mathcal{C} = \{0, \dots, 2^{k+1} - 1\} \times \{0, \dots, 2^{k+1} - 1\}.$$

2. Der Verschlüsselungsraum ist

$$\mathcal{K}_E = \{(q, g, g^a \mod q) \mid q \in \mathbb{P}, \text{size}(q) = k + 1,\ \langle [g] \rangle = \mathbb{Z}_q^\times,\ a \in \{2, \dots, q - 2\}\}.$$

3. Der Entschlüsselungsraum ist $\mathcal{K}_D = \{(q, g, a) \mid (q, g, g^a \mod q) \in \mathcal{K}_E, 2 \le a \le q - 2\}$.

4. Die zu $(q, g, A) \in \mathcal{K}_E$ gehörige Verschlüsselungsfunktion ist gegeben durch

$$E_{(q,g,A)}(p) = (g^b \mod q,\ A^b p \mod q),$$

wobei $b \in \{2, \dots, q - 2\}$ zufällig und gleichverteilt gewählt wird.

5. Die zu $(q, g, a) \in \mathscr{K}_D$ gehörige Entschlüsselungsfunktion ist gegeben durch

$$D_{(q,g,a)}(c_1, c_2) = c_1^{-a} c_2 \mod q. \qquad \triangleleft$$

Bemerkung 3.29 Bei der Verschlüsselung eines Klartexts $p \in \mathscr{P}$ zu

$$E_{(q,g,A)}(p) = (g^b \mod q,\ A^b p \mod q)$$

wird der Exponent $b \in \{2, \dots, q-2\}$ zufällig gewählt. Der Chiffretext hängt somit nicht nur vom Klartext und vom gewählten Schlüssel ab, sondern auch von der zufälligen Wahl von b. Dadurch wird das ElGamal-System zu einem probabilistischen Kryptosystem. $\triangleleft$

Beispiel 3.30 Seien $q = 23 \in \mathbb{P}$ und $g = 7$. Man kann nachrechnen, dass $\langle [7]_{23} \rangle = \mathbb{Z}_{23}^\times$ gilt. Wir wählen den Entschlüsselungsschlüssel $d = (q, g, a) = (23, 7, 6) \in \mathscr{K}_D$. Dann gilt

$$g^a \equiv 7^6 \equiv (7^2)^3 \equiv 49^3 \equiv 3^3 \equiv 4 \mod 23.$$

Der zu $d \in \mathscr{K}_D$ gehörige Verschlüsselungsschlüssel ist also $e = (q, g, g^a \mod q) = (23, 7, 4)$.

Wir verschlüsseln den Klartext $p = 7 \in \mathscr{P}$ für verschiedene $b \in \{2, \dots, 21\}$:

- $b = 3$: $E_{(23,7,4)}(7) = (7^3 \mod 23,\ 4^3 \cdot 7 \mod 23) = (21, 11) \in \mathscr{C}$
- $b = 2$: $E_{(23,7,4)}(7) = (7^2 \mod 23,\ 4^2 \cdot 7 \mod 23) = (3, 20) \in \mathscr{C}$

Es gilt dann:

- $D_{(23,7,6)}(21, 11) \equiv 21^{-6} \cdot 11 \equiv 11^6 \cdot 11 \equiv 6^3 \cdot 11 \equiv 9 \cdot 11 \equiv 99 \equiv 7 \mod 23$
- $D_{(23,7,6)}(3, 20) \equiv 3^{-6} \cdot 20 \equiv 8^6 \cdot 20 \equiv (-5)^3 \cdot 20 \equiv 13 \cdot 20 \equiv 260 \equiv 7 \mod 23$ $\triangleleft$

Satz 3.31

1. *Unter der Logarithmusannahme ist das ElGamal-Kryptosystem ein Public-Key-Kryptosystem.*
2. *Unter der Diffie-Hellman-Annahme erfüllt das ElGamal-Kryptosystem die Einwegeigenschaft.*

$\triangleleft$

Beweis.

1. Wir überprüfen die in der Definition eines Public-Key-Kryptosystems aufgeführten Eigenschaften:

   a) Seien $(q, g, g^a \mod q) \in \mathscr{K}_E$ und $(q, g, a) \in \mathscr{K}_D$. Für alle $p \in \mathscr{P}$ und alle (zufällig gewählten) $b \in \{2, \dots, q-2\}$ gilt

$$\begin{aligned} D_{(q,g,a)}(E_{(q,g,g^a)}(p)) &\equiv D_{(q,g,a)}(g^b \mod q, (g^a)^b p \mod q) \\ &\equiv (g^b)^{-a} \cdot (g^a)^b p \equiv g^{-ab+ab} p \\ &\equiv p \mod q. \end{aligned}$$

   Somit ist $D_{(q,g,a)}$ die zu $E_{(q,g,g^a)}$ gehörige Entschlüsselungsfunktion.

b) Die Ver- und Entschlüsselungsfunktionen haben polynomielle Laufzeit: Bei der Verschlüsselung ist sowohl das Berechnen von Potenzen $g^b \mod q$ und $A^b \mod q$ als auch die Multiplikation $A^b \cdot p \mod q$ in Polynomialzeit möglich. Dasselbe gilt für die Entschlüsselung; die Potenz $c_1^{-a} \mod q$ und das Produkt $c_1^{-1} \cdot c_2$ können in Polynomialzeit bestimmt werden.

c) Unter der Logarithmusannahme existiert kein polynomieller Algorithmus, mit dem $(q, g, a) \in \mathcal{K}_D$ aus $(q, g, A = g^a \mod q) \in \mathcal{K}_E$ berechnet werden kann:

Gäbe es einen Algorithmus der (q, g, a) aus (q, g, A) berechnet, dann kann aus der Gleichung

$$[g]^a = [A] \in \mathbb{Z}_q$$

der diskrete Logarithmus a von A zur Basis g bestimmt werden. Dies ist ein Widerspruch zur Logarithmusannahme.

2. Wir nehmen an, dass die Einwegeigenschaft nicht erfüllt ist, d. h. ein Angreifer kann Chiffretexte in polynomieller Zeit entschlüsseln. Dann kann er aus dem Schlüssel $(q, g, A) \in \mathcal{K}_E$ und einem Chiffretext $(g^b, g^{ab}p) \in \mathcal{C}$ den Klartext $p \in \mathcal{P}$ in polynomieller Zeit berechnen.

Insbesondere kann der Angreifer $(g^b, 1) \in \mathcal{C}$ entschlüsseln und erhält

$$D_{(q,g,a)}(g^b, 1) = (g^b)^{-a} \cdot 1 = g^{-ab} \mod q$$

als Klartext. Da $[g^{-ab}] \in \mathbb{Z}_q^\times$ in polynomieller Zeit invertiert werden kann, kann auch $[g^{ab}] \in \mathbb{Z}_q^\times$ in Polynomialzeit berechnet werden. Damit hat der Angreifer $g^{ab} \mod q$ aus $(q, g, g^a) \in \mathcal{K}_E$ und $(g^b, 1) \in \mathcal{C}$ in Polynomialzeit bestimmt. Ein Widerspruch zur Diffie-Hellman-Annahme. □

Bemerkung 3.32 1. Das ElGamal-Kryptosystem erfüllt die Nicht-Unterscheidbarkeit unter einem Chosen-Plaintext-Angriff, d. h. gegeben ein Chiffretext $c \in \mathcal{C}$ und eine Menge von Klartexten $\{p_1, \dots, p_n\}$ kann nicht in Polynomialzeit herausgefunden werden, welcher der Klartexte zum Chiffretext c verschlüsselt wurde:

Das ElGamal-System ist probabilistisch. Bei der Verschlüsselung von $p_i \in \{p_1, \dots, p_n\}$ mit $(q, g, A) \in \mathcal{K}_E$ wird ein zufälliges $b \in \{2, \dots, q-2\}$ gewählt. Jeder Klartext p_i entspricht also $q - 3$ möglichen Chiffretexte

$$(g^2 \mod q, A^2 p_i \mod q), \dots, (g^{q-2} \mod q, A^{q-2} p_i \mod q).$$

Um alle diese möglichen Chiffretexte zu berechnen und mit dem Chiffretext c zu vergleichen, müssen

$$q - 3 = \mathcal{O}\left(2^{\text{size}(q)}\right)$$

Berechnungen durchgeführt werden. Insbesondere ist die Rechenzeit hierfür nicht polynomiell in $\text{size}(q)$.

2. Die Nicht-Modifierzierbarkeit unter einem Chosen-Plaintext-Angriff ist verletzt:

   Ein Chiffretext
   $$c = E_{(q,g,A)}(p) = (g^b, A^b p) \in \mathscr{C}$$
   sei bekannt, aber nicht der zughehörige Klartext $p \in \mathscr{P}$. Für jeden Klartext $p' \in \mathscr{P}$ können wir den Chiffretext so verändern, dass er den Klartext $p' \cdot p \mod q$ verschlüsselt. Dazu definieren wir den neuen Chiffretext
   $$c' = (g^{b'} \cdot g^b \mod q, A^{b'} p' \cdot A^b p \mod q) = (g^{b'+b} \mod q,\ A^{b'+b} p' p \mod q).$$
   Dieser Chiffretext c' wird zu $p' \cdot p \mod q$ entschlüsselt:
   $$D_{(q,\ g,\ a)}(c') \equiv D_{(q,g,a)}(g^{b'+b} \mod q,\ A^{b'+b} p' p \mod q) \equiv p' p \mod q \qquad \triangleleft$$

3.3. Primzahltests

Für die vorgestellten Public-Key-Systeme werden große Primzahlen benötigt: zwei geheime $q_1, q_2 \in \mathbb{P}$ für das RSA-Verfahren, um $n = q_1, q_2$ zu bestimmen, und eine öffentliche $q \in \mathbb{P}$ für das ElGamal-Kryptosystem als Teil des Schlüssels $(q, g, A) \in \mathscr{K}_E$. Bisher haben wir so getan, als ob diese Primzahlen einfach vorliegen. Beim Gebrauch des Kryptosystems in der Praxis müssen sie jedoch erst einmal erzeugt werden.

Die Standardverfahren zur Primzahlerzeugung gehen wie folgt vor: Man erzeugt solange zufällige Zahlen in der gesuchten Größenordnung, bis eine Primzahl gefunden wird – oder eher: bis man glaubt, eine Primzahl gefunden zu haben. Um zu testen, ob eine Zahl eine Primzahl ist, werden zumeist probabilistische Algorithmen angewandt. Mit diesen probabilistischen Algorithmen kann nicht sicher festgestellt werden, ob eine Zahl eine Primzahl ist. Es kann lediglich festgestellt werden, dass eine Zahl mit einer gewissen Wahrscheinlichkeit eine Primzahl ist.

Bis heute sind keine effizienten deterministischen Algorithmen für Primzahltests bekannt. Erst 2002 wurde der erste polynomielle deterministische Primzahltest veröffentlicht, der sogenannte AKS-Test. Wegen der großen Exponenten ist seine Laufzeit für Praxisanwendungen aber immer noch zu groß.

3.3.1. Fermat-Test

Das Fermat-Test macht sich den kleinen Satz von Fermat zunutze, um zu testen, ob eine Zahl $n \in \mathbb{N}$ eine Primzahl ist. Dieser Satz besagt, dass $a^{q-1} = [1] \in \mathbb{Z}_q$ für alle Primzahlen $q \in \mathbb{P}$ und alle $a \in \mathbb{Z}_q^\times = \mathbb{Z}_q \setminus \{[0]\}$ gilt.

Definition 3.33 (Carmichael-Zahl) Sei $a, n \in \mathbb{N}$ mit $0 < a < n$.

1. n heißt *Pseudoprimzahl zur Basis a*, wenn $a^{n-1} \equiv 1 \mod n$ gilt.

2. Wenn $a^{n-1} \not\equiv 1 \mod n$ gilt, dann heißt *a Fermat-Zeuge gegen die Primalität von n.*

3. n heißt *Carmichael-Zahl*, wenn n keine Primzahl ist, aber auch keinen Fermat-Zeugen hat. ◁

Bemerkung 3.34 Insbesondere sind Primzahlen auch Pseudoprimzahlen zu allen Basen $0 < a < n$. ◁

Konstruktion 3.35 (Fermat-Test) Um von einer Zahl $n \in \mathbb{N}$ mit dem Fermat-Test zu überprüfen, ob sie eine Primzahl ist, berechnen wir für alle $0 < a < n$ die Zahl $a^{n-1} \mod n$. Gilt jeweils

$$a^{n-1} \equiv 1 \mod n,$$

dann besteht n den Fermat-Test. In diesem Fall wissen wir, dass n entweder eine Primzahl oder aber eine Carmichael-Zahl ist. Besteht n den Fermat-Test nicht, so wissen wir, dass n keine Primzahl (und auch keine Carmichael-Zahl) ist. ◁

Beispiel 3.36 Sei $n = 341$. Dann gilt $2^{340} \equiv 1 \mod 341$ und $3^{340} \equiv 56 \mod 341$. Somit ist 341 keine Primzahl, die Zahl 3 ist ein Fermat-Zeuge gegen die Primalität von 341. ◁

Bemerkung 3.37 Im Jahr 1994 wurde bewiesen, dass es unendlich viele Carmichael-Zahlen gibt, d. h. unendlich viele Zahlen, die keine Primzahlen sind, aber auch keine Fermat-Zeugen gegen ihre Primalität besitzen. Z. B. sind 561, 1105, 1729 die kleinsten Carmichael-Zahlen.

Somit können wir dem Fermat-Test nicht vertrauen und er ist für die Praxis ungeeignet. Besser geeignet ist der Miller-Rabin-Test, den wir im nächsten Abschnitt vorstellen. ◁

3.3.2. Miller-Rabin-Test

Der Miller-Rabin-Test ist ein probabilistischer Primzahltest. Mit diesem Test kann angegeben werden, wie wahrscheinlich es ist, dass eine Zahl $n \in \mathbb{N}$ eine Primzahl ist. Er macht sich folgendes Lemma zunutze.

Lemma 3.38 (Miller-Rabin) *Sei $q \in \mathbb{P}$ und $a \in \mathbb{Z}_q^\times$. Wir wählen $s, t \in \mathbb{N}$ so, dass $q - 1 = 2^s t$ mit t ungerade gilt. Dann folgt:*

$$a^t = [1]_q$$
$$\textit{oder} \qquad a^{2^r t} = -[1]_q \quad \textit{für ein } 0 \le r < s. \qquad ◁$$

Beweis. Sei $0 \le \overline{s} \le s$ minimal, so dass $a^{2^{\overline{s}} t} = [1]_q$ gilt. Wegen $a^{q-1} = a^{2^s t} = [1]_q$ existiert solch ein $\overline{s}$. Wir unterscheiden zwei Fälle:

1. Es gilt $\overline{s} = 0$: Dann erhalten wir $a^{2^{\overline{s}} t} = a^{2^0 t} = a^t = [1]_q$.

2. Es gilt $0 < \overline{s}$: Setze $r = \overline{s} - 1 < s$. Dann gilt

$$a^{2^{\overline{s}}t} = a^{2^{r+1}t} = (a^{2^r t})^2 = [1]_q.$$

Da $b^2 = [1]_q$ nur die Lösungen $b = \pm[1]_q \in \mathbb{Z}_q$ hat, folgt $a^{2^r t} = \pm[1]_q$. Da $\overline{s} > r$ minimal gewählt war mit der Eigenschaft, dass $a^{2^{\overline{s}}t} = [1]_q$ gilt, ergibt sich $a^{2^r t} = -[1]_q$. □

Mindestens eine der beiden Bedingung im obigen Lemma ist erfüllt, wenn q eine Primzahl ist. Andernfalls hat man einen Zeugen gegen die Primalität von n gefunden:

Definition 3.39 (Miller-Rabin-Zeuge) Sei $n \in \mathbb{N}$ und schreibe $n - 1 = 2^s t$ mit t ungerade (wie im obigen Lemma). Dann heißt $a \in \mathbb{Z}_n^\times$ *Miller-Rabin-Zeuge gegen die Primalität von n*, wenn gilt:

$$a^t \neq [1]_n \tag{3.2}$$

$$\text{und} \qquad a^{2^r t} \neq -[1]_n \quad \text{für alle } 0 \leq r < s. \tag{3.3}$$

◁

Beispiel 3.40 Sei $n = 561$ mit $n-1 = 560 = 2^4 \cdot 35$. Also ist $s = 4$ und $t = 35$. Für $a = [2] \in \mathbb{Z}_n$ berechnen wir

$$\begin{aligned} [2]_n^{1\cdot 35} &= [263]_n \\ [2]_n^{2\cdot 35} &= [166]_n \\ [2]_n^{2^2\cdot 35} &= [67]_n \\ [2]_n^{2^3\cdot 35} &= [1]_n \end{aligned}$$

Somit gilt $a^t \neq 1 \mod q$ und $a^{2^r t} \neq -1 \mod q$ für alle $0 \leq r < s = 4$. Folglich ist $[a] = [2] \in \mathbb{Z}_n^\times$ ein Miller-Rabin-Zeuge gegen die Primalität der Carmichael-Zahl 561. ◁

Beispiel 3.41 Wir bestimmen alle Miller-Rabin-Zeugen gegen die Primalität von $n = 15$:

Es gilt $n - 1 = 14 = 2^1 \cdot 7$, also $s = 1$ und $t = 7$. $a \in \mathbb{Z}_n^\times$ ist genau dann Miller-Rabin-Zeuge von 15, wenn $a^7 \neq \pm[1]_{15}$ (denn $a^t = a^7 = a^{2^0 7} = a^{2^{s-1}t}$).

a	1	2	4	7	8	11	13	14
$a^7 \mod 15$	1	8	4	13	2	11	7	14

Also sind alle Elemente von $\mathbb{Z}_{15}^\times$ außer $[1]$ und $[14] = -[1]$ Zeugen gegen die Primalität von 15. ◁

Das obige Beispiel zeigt im Fall $n = 15$, dass die „meisten“ $a \in \mathbb{Z}_n^\times$ Zeugen gegen die Primalität von 15 sind. Das folgende Lemma, das wir ohne Beweis angeben, besagt, dass diese Aussage nicht nur für 15, sondern für alle Zahlen $n \geq 3$ mit $2, 3 \nmid n$ erfüllt ist.

Satz 3.42 *Sei $n \geq 3$ keine Primzahl und es gelte $2, 3 \nmid n$. Wir definieren*

$$\overline{MRZ}(n) := \{a \in \mathbb{Z}_n^\times \mid a \underline{\text{kein}} \text{ Miller-Rabin-Zeuge gegen die Primalität von } n\}.$$

Dann gilt

$$\left|\overline{MRZ}(n)\right| \leq \varphi(n)/4 < n/4. \qquad \triangleleft$$

Konstruktion 3.43 (Miller-Rabin-Test) Wir testen, ob $2^s t = n \in \mathbb{N}$ eine Primzahl ist (t ist wieder eine ungerade Zahl). Dazu wählen wir $i \in \mathbb{N}$ zufällige $a \in \mathbb{Z}_n^\times$ und berechnen

$$a^t, \ldots, a^{2^{s-1}t} \in \mathbb{Z}_n^\times.$$

Finden wir einen Miller-Rabin-Zeugen gegen die Primalität von n, d. h. gilt $a^t \neq [1]_n$ und $a^{2^r t} \neq -[1]_n$ für alle $0 \leq r < s$, so ist n wegen Lemma 3.38 keine Primzahl. Finden wir keinen Miller-Rabin-Zeugen, so ist gemäß Satz 3.42 die Wahrscheinlichkeit dafür, dass n keine Primzahl ist, kleiner als

$$\left(\frac{1}{4}\right)^i.$$

Beispielsweise ist für $i = 10$ (d. h. von 10 Zahlen wurde getestet, ob sie Miller-Rabin-Zeugen sind) die Fehlerwahrscheinlichkeit des Miller-Rabin-Tests kleiner als

$$\left(\frac{1}{4}\right)^{10} = \frac{1}{16^5} < \frac{1}{10^5}.$$

Weiterführende Analysen haben ergeben, dass die Fehlerwahrscheinlichkeit des Miller-Rabin-Tests sogar noch geringer ist. $\triangleleft$

Bemerkung 3.44 Aus der (bisher nicht bewiesenen) verallgemeinerten Riemannschen Vermutung folgt, dass n eine Primzahl ist, wenn die Bedingung aus Lemma 3.38 für alle $[a] \in \mathbb{Z}_n$ mit $1 < a \leq 2\ln^2(n)$ erfüllt ist. Bisher konnte diese Vermutung für $n < 3{,}4 \cdot 10^{14}$ bewiesen werden. Stellt sich die verallgemeinerte Riemannsche Vermutung als wahr heraus, erhält man durch wiederholte Durchführung des Miller-Rabin-Tests für alle $1 \leq a \leq 2\ln^2(n)$ ein deterministischen Primzahltest. $\triangleleft$

4. Über Verschlüsselung hinaus

Die bisher vorgestellten kryptographischen Verfahren garantieren die Vertraulichkeit einer Nachricht: Der Absender einer Nachricht möchte verhindern, dass ein anderer als der vorgesehene Empfänger die Nachricht entschlüsseln kann. In diesem Kapitel werden wir untersuchen, wie weitere kryptographische Ziele erreicht werden können.

4.1. Digitale Signaturen

Digitale Signaturen haben eine ähnliche Funktion wie gewöhnliche Unterschriften. Wenn ein Dokument signiert wurde, kann jeder, der das Dokument sieht und die Unterschrift des Verfassers kennt, überprüfen, ob das Dokument tatsächlich vom Verfasser stammt. Das nennt man *Authentizität*. Da eine Unterschrift z. B. auf dem Personalausweis gespeichert ist, kann durch eine Unterschrift des Weiteren Dritten gegenüber nachgewiesen werden, wer der Verfasser ist. Das Dokument wird durch die Unterschrift *zurechenbar*.

Digitale Signaturen erfüllen über *Zurechenbarkeit* hinaus eine weitere Funktion. Durch sie kann überprüft werden, ob das Dokument nachträglich von jemand anderem als dem Verfasser verändert wurde. Diese Eigenschaft nennt man *Integrität*.

Definition 4.1 (Digitale Signatur) Eine *digitale Signatur* ist ein 5-Tupel

$$(\mathcal{M},\ \mathcal{U},\ \mathcal{K} = \mathcal{K}_S \dot{\cup} \mathcal{K}_V,\ \mathcal{S},\ \mathcal{V})$$

mit

1. Menge der Nachrichten $\mathcal{M}$,
2. Menge der Unterschriften $\mathcal{U}$,
3. Schlüsselmenge $\mathcal{K}$, die in die Menge der geheimen *Signierschlüssel* $\mathcal{K}_S$ und die Menge der öffentlichen *Verifikationsschlüssel* $\mathcal{K}_V$ partitioniert ist,
4. Menge der *Signierfunktionen* $\mathcal{S} = \{S_s : \mathcal{M} \to \mathcal{U} \mid s \in \mathcal{K}_S\}$ und
5. Menge der *Verifikationsfunktionen* $\mathcal{V} = \{V_v : \mathcal{M} \times \mathcal{U} \to \{0,1\} \mid v \in \mathcal{K}_V\}$,

so dass für jeden *Signierschlüssel* $s \in \mathcal{K}_S$ ein *Verifikationsschlüssel* $v(s) \in \mathcal{K}_V$ existiert, für den

$$V_{v(s)}(m,u) = \begin{cases} 1 & \text{falls } u = S_s(m) \\ 0 & \text{sonst} \end{cases}$$

für alle Nachrichten $m \in \mathcal{M}$ und alle Unterschriften $u \in \mathcal{U}$ gilt. Sowohl für die Auswertung der Signierfunktionen als auch für die Auswertung der Verifikationsfunktionen existieren polynomielle Algorithmen. ◁

Beispiel 4.2 Alice schließt einen Vertrag übers Internet ab. Ihr Vertragspartner Bob möchte später nachweisen können, dass er einen Vertrag mit Alice (und nicht mit jemand anderem) in dieser Form geschlossen hat. Es wird eine digitale Signatur genutzt:

Alice hat einen geheimen Signierschlüssel $s \in \mathcal{K}_S$ gewählt und sicher gespeichert. Den zugehörigen Verifikationsschlüssel $v(s) \in \mathcal{K}_V$ macht sie öffentlich zugänglich: Der Verifikationsschlüssel $v(s)$ liegt in einem Verzeichnis, auf das jeder zugreifen kann und das nicht verändert werden kann. Der Vertrag $m \in \mathcal{M}$ wird nun durch $u = S_s(m)$ signiert.

Möchte Bob nachprüfen bzw. einem Dritten gegenüber nachweisen, dass der Vertrag m tatsächlich in dieser Form von Alice (und nicht von jemand anderem oder überhaupt nicht) geschlossen wurde, greift er auf den öffentlichen Verifikationsschlüssel $v(s) \in \mathcal{K}_V$ zu und berechnet

$$V_{v(s)}(m,u).$$

Kommt $V_{v(s)}(m,u) = 1$ heraus, muss gemäß der Definition einer digitalen Signatur 4.1 die Gleichheit $u = S_s(m)$ gelten. Kommt $V_{v(s)}(m,u) = 0$ heraus, so wurde der Vertrag m *nicht* mit s zu u signiert.

Bob und jeder andere kann somit überprüfen, ob der Vertrag m mit Alice' privatem Signierschlüssel s unterzeichnet wurde. Ist die Unterschrift $u = S_s(m)$ fälschungssicher (d. h. kann niemand, der $s \in \mathcal{K}_S$ nicht kennt, $u = S_s(m)$ berechnen und kann kein anderer Vertrag m', der $S_s(m') = u$ erfüllt, erzeugt werden), kann Bob einer dritten Partei gegenüber nachweisen, dass Alice (und niemand anderes) den Vertrag *in dieser Form* unterzeichnet hat. ◁

Normalerweise werden nicht ganze Nachrichten, sondern nur sogenannte Hashwerte von Nachrichten signiert. Hashfunktionen führen wir später ein. Erst einmal stellen wir Angriffe auf digitale Signaturen vor.

Definition 4.3 (Angriffe auf digitale Signaturen)

1. *Key-Only-Angriff (KOA)*: Der Angreifer kennt den öffentlichen Verifikationsschlüssel des Unterzeichners.
2. *Known-Message-Angriff (KMA)*: Der Angreifer erhält einige (nicht selbst gewählte) Nachrichten und die zugehörigen Signaturen.
3. *Chosen-Message-Angriff (CMA)*: Der Angreifer kann beliebige Nachrichten signieren lassen und erhält die Signatur. ◁

Definition 4.4 (Ziele beim Angriff auf digitale Signaturen) Die folgenden Ziele werden beim Angriff auf eine digitale Signatur unterschieden:

1. *Total Break*: Der geheime Signierschlüssel $s \in \mathcal{K}_S$ wird aufgedeckt.
2. *Universelle Fälschung*: Für eine vorgegebene Nachricht $m \in \mathcal{M}$ kann die Signatur $u = S_s(m)$ gefälscht werden.
3. *Existentielle Fälschung*: Für einige (aber nicht für jede vorgegebene) Nachrichten $m \in \mathcal{M}$ kann die Signatur $u = S_s(m)$ gefälscht werden. ◁

Das stärkste Sicherheitsmodell ist Sicherheit gegen existentielle Fälschung unter einem Chosen-Message-Angriff.

4.1.1. RSA-Signatur

Aus Public-Key-Kryptosystemen, bei denen die Reihenfolge von Ver- und Entschlüsselung umgekehrt werden kann, können digitale Unterschriften konstruiert werden. Wir zeigen die Konstruktion beispielhaft für die RSA-Funktion.

Konstruktion 4.5 (RSA-Signatur) Die RSA-Signatur ist wie folgt definiert. Mit

$$(\mathcal{P}, \mathcal{C}, \mathcal{K}_E \dot{\cup} \mathcal{K}_D, \mathcal{E}, \mathcal{D})$$

bezeichnen wir das RSA-Kryptosystem.

1. $\mathcal{M} = \mathcal{P} = \{0, \ldots, 2^k - 1\}$, $\mathcal{U} = \mathcal{C} = \{0, \ldots, 2^{k+1} - 1\}$
2. $\mathcal{K}_S = \mathcal{K}_D$, $\mathcal{K}_V = \mathcal{K}_E$
3. Sei $s = (n, d, e) \in \mathcal{K}_S$. Dann ist $S_s : \mathcal{M} \to \mathcal{S},\ m \mapsto D_d(m) = m^d \mod n$.
4. Sei $v = (n, e) \in \mathcal{K}_V$. Dann ist

$$\mathcal{V}_v : \mathcal{M} \times \mathcal{S} \to \{0, 1\},\ (m, u) \mapsto \begin{cases} 1 & E_v(u) \equiv u^e \equiv m \mod n, \\ 0 & \text{sonst.} \end{cases}$$

◁

Bemerkung 4.6 Bei der RSA-Signatur wird im Vergleich zum RSA-Verfahren die Rolle von Ver- und Entschlüsselung gerade umgedreht:

Bei der RSA-Signatur wird eine Nachricht erst mit einem geheimen $s = (n, d, e) \in \mathcal{K}_S = \mathcal{K}_D$ zur Unterschrift $u = m^d \mod n$ „entschlüsselt“, so dass danach jeder die Unterschrift u mithilfe des öffentlichen Verifikationsschlüssels $v(s) = (n, e) \in \mathcal{K}_V = \mathcal{K}_E$ wieder zu

$$E_{v(s)}(u) \equiv u^e \equiv (m^d)^e \equiv m \mod n$$

„verschlüsseln“ kann. ◁

Bemerkung 4.7 Es folgt aus 3.21, dass die RSA-Signatur die in der Definition einer digitalen Signatur gelisteten Eigenschaften erfüllt. ◁

Bemerkung 4.8 (Sicherheitseigenschaften der RSA-Signatur)

1. Unter der RSA-Annahme ist bei einem Key-Only-Angriff weder ein Total Break noch universelle Fälschung in polynomieller Zeit möglich:

   Das RSA-Verfahren ist unter der RSA-Annahme ein Public-Key-Kryptosystem. Daher kann der private Signierschlüssel $s = (n, d, e) \in \mathcal{K}_S = \mathcal{K}_D$ *nicht* in polynomieller Zeit aus dem zugehörigen öffentlichen Verifikationsschlüssel $v(s) = (n, e) \in \mathcal{K}_V = K_E$ berechnet werden. Somit ist kein Total Break in polynomieller Zeit möglich.

   Außerdem haben wir unter der RSA-Annahme gezeigt, dass $u = S_s(m) = D_s(m) = m^d \mod n$ nicht in polynomieller Zeit aus $v(s) = (n, e)$ und $m \in \mathcal{M}$ berechnet werden kann. Somit ist universelle Fälschung nicht in polynomieller Zeit möglich.

2. Existentielle Fälschung, d. h. Fälschung einiger Unterschriften, ist unter einem Key-Only-Angriff möglich:

   Der öffentliche Schlüssel $v(s) = (n, e) \in \mathcal{K}_V$ ist bekannt, der zugehörige private Schlüssel $s = (n, d, e) \in \mathcal{K}_S$ bleibt geheim. Nun kann für jede Unterschrift $u \in \mathcal{U}$ die Nachricht $m = E_{v(s)}(u) = u^e \mod n$ berechnet werden. Diese Nachricht m hat die Unterschrift

   $$S_s(m) \equiv m^d \equiv (u^e)^d \equiv u \mod n.$$

   Somit kann die Unterschrift $u \in \mathcal{U}$ der Nachricht $m = E_{v(s)}(u) \in \mathcal{M}$ gefälscht werden, ohne den privaten Schlüssel $s \in \mathcal{K}_S$ zu kennen. Man beachte, dass nur die Unterschrift u, aber nicht die signierte Nachricht m frei gewählt weden kann!

3. Die Signierfunktionen $S_s \in \mathcal{S}$ sind multiplikativ. Daher ist universelle Fälschung unter einem Chosen-Message-Angriff möglich:

   Um $m \in \mathcal{M}$ mit dem Signierschlüssel $s = (n, d, e) \in \mathcal{K}_S$ zu signieren (ohne den Schlüssel s zu kennen), zerlegt man m in ein Produkt $m = m_1 \cdot m_2$ mit $m_1, m_2 \neq m$. Wegen $m_1, m_2 \neq m$ können bei einem Chosen-Message-Angriff die beiden Signaturen $S_s(m_i)$, $i = 1, 2$, angefordert werden. Diese können wegen der Multiplikativität der Signierfunktion $S_s \in \mathcal{S}$ ohne Kenntnis von s zur Signatur von m

   $$S_s(m) \equiv S_s(m_1 \cdot m_2) \equiv (m_1 \cdot m_2)^d \equiv m_1^d \cdot m_2^d \equiv S_s(m_1) \cdot S_s(m_2) \mod n$$

   aneinandergereiht werden. ◁

4.1.2. Hashfunktionen

Bei der vorgestellten RSA-Signatur hat die Unterschrift dieselbe Länge wie die Nachricht. Das ist ineffizient, da zum einen die lange Unterschrift erst einmal erzeugt werden muss und sie dann zum anderen vergleichsweise viel Speicherplatz verbraucht. Mithilfe von *Hashfunktionen* können kürzere digitale Signaturen erzeugt werden.

Definition 4.9 (Hashfunktion) Eine Funktion $H : \mathbb{N} \to \{0, \dots, 2^l - 1\}$ mit $l \in \mathbb{N}$ heißt *Hashfunktion*, wenn sie in Polynomialzeit berechnet werden kann. ◁

Bemerkung 4.10 Der Definitionsbereich einer Hashfunktion ist in der obigen Definition durch $\mathbb{N}$ gegeben, der Zielbereich durch $\{0, \dots, 2^l - 1\}$. Interpretiert in Binärdarstellung entspricht $\mathbb{N}$ der Menge aller endlichen Bitfolgen $\{0, 1\}^*$ (die mit 1 beginnen). Analog entspricht $\{0, \dots, 2^l - 1\}$ der Menge aller Bitfolgen $\{0, 1\}^l$ der festen Länge l. Auf diese Weise interpretiert kann eine Hashfunktion als eine Funktion $H : \{0, 1\}^* \to \{0, 1\}^l$ angesehen werden, die deren Auswertung in Polynomialzeit möglich ist. Anschaulich gesprochen formt eine Hashfunktion eine Bitfolge beliebiger Länge (z. B. eine Nachricht) in vertretbarer Zeit in eine Bitfolge der festen Länge l um. ◁

Beispiel 4.11 (Hashfunktionen aus Blockchiffren) Sei $(A, l, \mathcal{K}, \mathcal{E})$ eine Blockchiffre mit Alphabet $A = \{0, 1\}$, Blocklänge $l \in \mathbb{N}$, Schlüsselraum $\mathcal{K} = A^l$ und Verschlüsselungsfunktionen $E_e : A^l \to A^l$, $e \in \mathcal{K}$. Dann ist durch die folgende Rekursion eine Hashfunktion $H : \{0, 1\}^* \to \{0, 1\}^l$ definiert:

- $H(\emptyset) = 0 \dots 0 \in \{0, 1\}^l$
- $H(x_1, \dots, x_r) = E_h(x_r) + h$ für alle $(x_1, \dots, x_r) \in (A^l)^r$ mit $r > 0$ und $h = H(x_1, \dots, x_{r-1})$

Konkreter: Als Blockchiffre der Länge 4 auf dem Alphabet $\{0, 1\}$ wählen wir das One-Time-Pad. Es gilt $\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0, 1\}^4$. Wir konstruieren den Hashwert von $x = 1101\ 0011\ 0010$ unter der zum One-Time-Pad gehörigen Hashfunktion $H : \{0, 1\}^* \to \{0, 1\}^4$:

- $H(\emptyset) = 0000$
- $H(1101) = E_{0000}(1101) + H(\emptyset) = (0000 + 1101) + 0000 = 1101$
- $H(1101\ 0011) = E_{1101}(0011) + H(1101) = (1101 + 0011) + 1101 = 0011$
- $H(1101\ 0011\ 0010) = E_{0011}(0010) + H(1101\ 0011) = (0011 + 0010) + 0011 = 0010$

Die weit verbreitete Hashfunktion SHA-1 (engl.: secure hash algorithm) basisert auf diesem Konzept. SHA-1 wird heutzutage jedoch nicht mehr als sicher eingestuft und wird nach und nach von SHA-2 bzw. SHA-3 abgelöst. SHA-3 ist 2012 als Sieger aus einem vom NIST veranstalteten Wettbewerb hervorgegangen, vergleichbar zu AES. ◁

Hashfunktionen werden für kryptographische Zwecke interessant, wenn sie kollisionsresistent sind.

Definition 4.12 Ein Pärchen $(x_1, x_2) \in \mathbb{N} \times \mathbb{N}$ heißt *Kollision der Hashfunktion* $H : \mathbb{N} \to \{0, \dots, 2^l - 1\}$, wenn $x_1 \neq x_2$ und $H(x_1) = H(x_2)$ gilt. Die Hashfunktion H heißt *kollisionsresistent*, wenn es keinen polynomiellen Algorithmus gibt, der eine Kollision $(x_1, x_2) \in \mathbb{N} \times \mathbb{N}$ von H berechnet. ◁

Bemerkung 4.13 Eine kollisionsresistente Hashfunktion $H : \mathbb{N} \to \{0, \dots, 2^l - 1\}$ ist eine Einwegfunktion:

Wähle ein $x \in \mathbb{N}$ und berechne $h = H(x)$. Angenommen H wäre keine Einwegfunktion, dann könnte in Polynomialzeit ein Urbild $y \in \mathbb{N}$ von $h \in \{0, \dots, 2^l - 1\}$ unter H berechnet werden. Im Fall $x \neq y$ hätte man eine Kollision (x, y) gefunden.

Da nicht bekannt ist, ob Einwegfunktionen existieren, ist auch nicht bekannt, ob kollisionsresistente Hashfunktionen existieren. Ihre Existenz kann nur unter Annahmen gezeigt werden. ◁

Da die Konstruktion kollisionsresistenter Hashfunktionen recht technisch ist, geben wir sie nur in einem Exkurs an. Stattdessen gehen wir auf Anwendungen kollisionsresistenter Hashfunktionen ein.

Bemerkung 4.14 (Authentizität) Sei $\mathcal{K}$ eine Schlüsselmenge. Eine Familie $\{H_k \mid k \in \mathcal{K}\}$ kollisionsresistenter Hashfunktionen heißt *Message Authentification Code (MAC)*. Ein Message Authentification Code erlaubt es, die Authentizität einer Nachricht zu überprüfen:

Alice möchte eine Nachricht $m \in \mathbb{N}$ an Bob schicken und sicher gehen, dass die Nachricht auf dem Weg zu Bob nicht verändert wird. Dazu tauscht sie einen geheimen Schlüssel $k \in \mathcal{K}$ mit Bob aus. Nun hasht sie ihre Nachricht zu $h = H_k(m)$ und schickt das Tupel (m, h) an Bob. Bob überprüft die Authentizität der Nachricht (m, h), indem er ebenfalls $H_k(m)$ berechnet und $H_k(m) = h$ überprüft.

Angenommen die Nachricht $(m, H_k(m))$ wird von Eve abgefangen und Eve möchte Bob glauben lassen, dass die Nachricht eigentlich $m' \in \mathbb{N}$ ist. Um Bob zu überzeugen, müsste Eve ein Pärchen $(m', H_k(m'))$ erzeugen. Da der Schlüssel $k \in \mathcal{K}$ aber geheim ist, kann sie die Unterschrift $H_k(m')$ nicht berechnen. Und da die Hashfunktion H_k kollisionsresistent ist, kann sie in polynomieller Zeit keine weitere Nachricht $m \neq m' \in \mathbb{N}$ erzeugen, die den Hashwert $H_k(m') = H_k(m)$ besitzt. ◁

Bemerkung 4.15 (Identifikation) Die Überprüfung einer Zugangsberechtigung, beispielsweise beim Einloggen fürs Online-Banking, wird auch Identifikation genannt. Dabei werden häufig Passwörter abgefragt. Nach der Eingabe des Passworts muss vom Anbieter des Dienstes überprüft werden, ob das Passwort korrekt ist. Um Missbrauch vorzubeugen, möchte man jedoch vermeiden, dass das Passwort vom Anbieter des Dienstes im Klartext gespeichert wird. Eine mögliche Vorgehensweise ist die folgende:

Vom Anbieter des Dienstes wird nicht das Passwort $w \in \{0, 1\}^*$, sondern das Bild des Passworts $f(w) \in \{0, 1\}^*$ unter einer Einwegfunktion $f : \{0, 1\}^* \to \{0, 1\}^*$ gespeichert. Um Passwörter beliebiger Länge zuzulassen und um gleichzeitig die Länge des Bildes $f(w)$ zu beschränken,

wird als Einwegfunktion häufig eine kollisionsresistente Hashfunktion $f : \{0,1\}^* \to \{0,1\}^l$ genutzt.

Der Nutzer gibt ein Passwort $w' \in \{0,1\}^*$ ein und es wird zuerst der Hashwert $f(w') \in \{0,1\}^l$ des Passworts berechnet. Dieser Hashwert $f(w')$ wird nun übertragen und mit dem gespeicherten Wert $f(w) \in \{0,1\}^l$ verglichen. Gilt $f(w) = f(w')$, nimmt man an, dass auch $w = w'$ gilt (denn die Berechnung einer Kollision (w, w') von f ist - zumindest unter Standardannahmen - nicht in Polynomialzeit möglich). In diesem Fall wird der Zugang erteilt. Gilt $f(w) \neq f(w')$, weiß man, dass auch $w \neq w'$ gilt, und der Zugang wird abgelehnt. ◁

Exkurs: Kollisionsresistente Hashfunktionen

Um kollisionsresistente Hashfunktionen zu konstruieren, werden sogenannte Kompressionsfunktionen genutzt.

Definition 4.16 Seien $l, m \in \mathbb{N}$. Eine Funktion $K : \{0, \dots, 2^m - 1\} \to \{0, \dots, 2^l - 1\}$ heißt Kompressionsfunktion, wenn $m > l$ gilt und wenn K in polynomieller Zeit ausgewertet werden kann. ◁

Wieder können wir Elemente von $\{0, \dots, 2^m - 1\}$ (bzw. $\{0, \dots, 2^l - 1\}$) als Bitfolgen der festen Länge $m \in \mathbb{N}$ (bzw. $l \in \mathbb{N}$) interpretieren. Anschaulich gesprochen wandelt eine Kompressionsfunktion eine Bitfolge der festen Länge m in eine kürzere Bitfolge der festen Länge l um.

Konstruktion 4.17 (Hashfunktionen aus Kompressionsfunktionen) Es sei eine Kompressionsfunktion $K : \{0,1\}^m \to \{0,1\}^l$ gegeben. Dann kann rekursiv eine Hashfunktion

$$H^K : \{0,1\}^* \to \{0,1\}^l$$

konstruiert werden:

Wir setzen $r = m - l > 0$ und zerlegen $x \in \{0,1\}^*$ in Bitfolgen der Länge r, also $x = (x_1, \dots, x_k)$ mit $x_1, \dots, x_k \in \{0,1\}^r$. Nun definieren wir rekursiv

- $H^K(\emptyset) = 0 \dots 0 \in \{0,1\}^l$,
- $H^K(x_1, \dots, x_k) = K\left(H^K(x_1, \dots, x_{k-1}) \circ x_k\right)$ für $k > 0$, wobei $\circ$ für die Verkettung von Wörtern steht.

  Man beachte: $H^K(x_1, \dots, x_{k-1})$ hat Länge l, x_k hat Länge r, also hat $y = H^K(x_1, \dots, x_{k-1}) \circ x_k$ Länge $m = l + r$ und $K : \{0,1\}^m \to \{0,1\}^l$ kann auf y angewandt werden. ◁

Beispiel 4.18 Wir betrachten die Kompressionsfunktion $K : \{0,1\}^8 \to \{0,1\}^4$, die für $(x_1, x_2) \in \{0,1\}^8$ mit $x_1, x_2 \in \{0,1\}^4$ durch

$$K(x_1, x_2) = x_1 + x_2$$

gegeben ist. Es gilt $r = 8 - 4 = 4$. Unter der zugehörigen Hashfunktion $H^K : \{0,1\}^* \to \{0,1\}^4$ berechnen wir das Bild von $x = 1101\ 0011\ 0010$:

- $H^K(\emptyset) = 0000$
- $H^K(1101) = K\left(H^K(\emptyset) \circ 1101\right) = K(000\,1101) = 1101$
- $H^K(1101\,0011) = K\left(H^K(1101) \circ 0011\right) = K(1101 \circ 0011) = 1110$
- $H^K(1101\,0011\,0010) = K\left(H^K(1101\,0011) \circ 0010\right) = K(1110 \circ 0010) = 1100.$ ◁

Wir geben folgenden Satz ohne Beweis an:

Satz 4.19 *Ist $K : \{0,\dots,2^m-1\} \to \{0,\dots,2^l-1\}$ eine kollisionsresistente Kompressionsfunktion, so ist auch die Hashfunktion $H^K : \mathbb{N} \to \{0,\dots,2^l-1\}$ kollisionsresistent.* ◁

Beispiel 4.20 (Kollisionsresistente Kompressionsfunktion) Sei $p \in \mathbb{P}$ eine Primzahl mit $\text{size}(p) = m+1 \in \mathbb{N}$. Seien außerdem $a, b \in \{1,\dots,2^m-1\}$ so gewählt, dass $[a] \in \mathbb{Z}_p^\times$ ein Erzeuger ist, d. h. es gilt

$$\langle [a] \rangle = \{[a]^0, [a]^1, [a]^2 \dots, \} = \mathbb{Z}_p^\times.$$

Wir definieren die Funktion

$$\begin{aligned} K_{a,b} : \{0,\dots,2^m-1\}^2 &\to \{0,\dots,2^{m+1}-1\} \\ (x,y) &\mapsto a^x b^y \mod p. \end{aligned}$$

Aus der Kompressionsfunktion $K_{a,b}$ kann rekursiv mit Konstruktion 4.17, eine Hashfunktion $H^{K_{a,b}}$ konstruiert werden. Unter der Logarithmusannahme ist die Kompressionsfunktion $K_{a,b}$, und somit mit Satz 4.19 auch die Hashfunktion $H^{K_{a,b}}$ kollisionsresistent:

Angenommen es wurde eine Kollision

$$K_{a,b}(x,y) \equiv a^x b^y \equiv a^{x'} b^{y'} \equiv K_{a,b}(x',y') \mod p$$

berechnet. Dann gilt $(x,y) \neq (x',y')$.

- Gilt $y = y'$, so folgt $a^x = a^{x'}$. Da $[a]$ die Gruppe $\mathbb{Z}_p^\times$ erzeugt und

  $$x,\ x' < 2^m < p$$

  gilt, folgt $x = x'$; ein Widerspruch dazu, dass $(x,y) \neq (x',y')$ eine Kollision von $K_{a,b}$ ist.
- Nehmen wir also $y \neq y'$ an. Da $[a] \in \mathbb{Z}_p^\times$ ein Erzeuger ist, existiert ein $z \in \mathbb{N}$, so dass $a^z \equiv b \mod p$ gilt. Dann erhalten wir:

  $$\begin{aligned} & a^x b^y \equiv a^{x'} b^{y'} \mod p \\ \Rightarrow\ & a^{x-x'} \equiv b^{y'-y} \equiv a^{z(y'-y)} \mod p \\ \Rightarrow\ & x - x' \equiv z(y'-y) \mod p-1 \end{aligned}$$

  Somit kann ein $z \equiv (x-x') \cdot (y'-y)^{-1} \mod p-1$, das $a^z \equiv b \mod p$ erfüllt, in Polynomialzeit berechnet werden; ein Widerspruch zur Logarithmusannahme. ◁

4.1.3. ElGamal-Signatur

Mithilfe kollisionsresistenter Hashfunktionen $H : \mathbb{N} \to \{0, \dots, 2^l - 1\}$ können Verallgemeinerungen digitaler Signaturen konstruiert werden. Diese ermöglichen es, die Länge der Unterschrift deutlich kürzer als die Länge der Nachricht zu halten. Des Weiteren kann durch die Nutzung von Hashfunktionen die Sicherheit der digitalen Signatur erhöht werden.

Wir stellen eine digitale Signatur mit Hashfunktion vor, die ElGamal-Signatur. Analog zum ElGamal-Kryptosystem ist die ElGamal-Signatur eine probabilistische Signatur: Die Unterschrift hängt nicht nur von der Nachricht und dem gewählten Schlüssel ab; bei der Berechnung der Unterschrift wird eine zufällige Wahl getroffen.

Konstruktion 4.21 (ElGamal-Signatur) Für $q \in \mathbb{P}$ prim mit size $q = k + 1 \in \mathbb{N}$ und für die (öffentliche) kollisionsresistente Hashfunktion $H : \mathbb{N} \to \{0, \dots, 2^k - 1\}$ ist die ElGamal-Signatur wie folgt definiert:

1. $\mathcal{M} = \mathbb{N}$ als Menge der Nachrichten
2. $\mathcal{U} = \{0, \dots, 2^{k+1} - 1\} \times \{0, \dots, 2^{k+1} - 1\}$ als Menge der Unterschriften
3. $\mathcal{K}_S = \{(g, a) \mid g, a \in \{2, \dots, q-2\}, \langle [g] \rangle = \mathbb{Z}_q^\times\}$ als Menge der Signierschlüssel
4. $\mathcal{K}_V = \{(g, g^a \mod q) \mid (g, a) \in \mathcal{K}_S\}$ als Menge der Verifikationsschlüssel
5. Sei $s = (g, a) \in \mathcal{K}_S$. Dann ist

   $$S_s : \mathcal{M} \to \mathcal{U}, \quad m \mapsto (r, t) = (g^b \mod q, b^{-1}(H(m) + aH(g^b \mod q)) \mod q - 1)$$

   für ein zufälliges $b \in \{2, \dots, q-2\}$ mit $\mathrm{ggT}(b, q-1) = 1$.
6. Sei $v = (g, A) \in \mathcal{K}_V$. Dann ist

   $$\mathcal{V}_v : \mathcal{M} \times \mathcal{S} \to \{0, 1\}, \quad (m, (r, t)) \mapsto \begin{cases} 1 & \text{falls } g^{H(m)} A^{H(r)} \equiv r^t \mod q \\ 0 & \text{sonst.} \end{cases}$$

◁

Lemma 4.22 *Die ElGamal-Signatur ist eine digitale Signatur.* ◁

Beweis. Zur Auswertung von V_v und S_s existieren polynomielle Algorithmen, da nur Produkte, Summen, Potenzen, Inverse von Elementen in $\mathbb{Z}_{q-1}$ und Hashwerte berechnet werden müssen.

Wir zeigen: Der zum Signierschlüssel $s = (g, a) \in \mathcal{K}_S$ gehörige Verifikationsschlüssel ist $v(s) = (g, A) \in \mathcal{K}_V$ mit $A = g^a \mod q$. Dazu überprüfen wir, dass $V_{v(s)}(m, u) = 1$ genau dann gilt, wenn $u = S_s(m)$ eine mögliche Unterschrift von m unter Nutzung des Signierschlüssels s ist.

1. Zuerst betrachten wir ein Tupel $(m, S_s(m)) \in \mathscr{M} \times \mathscr{U}$. Dann existiert ein $b \in \{2, \dots, q-2\}$ mit
$$S_s(m) = (g^b \mod q, b^{-1}(H(m) + aH(g^b \mod q)).$$
Mit der Notation $(r,t) \coloneqq S_s(m)$ folgt
$$r^t \equiv (g^b)^{b^{-1}(H(m)+aH(r))} \equiv g^{H(m)+aH(r)} \equiv g^{H(m)} A^{H(r)} \mod q,$$
und wir erhalten $V_{v(s)}(m, S_s(m)) = V_{v(s)}(m,(r,t)) = 1$.
2. Nun nehmen wir an, dass $V_{v(s)}(m,(r,t)) = 1$ gilt. Da g die Gruppe $\mathbb{Z}_q^\times$ erzeugt, existiert ein $b \in \mathbb{N}$, das $r \equiv g^b \mod q$ erfüllt. Es folgt
$$g^{bt} \equiv (g^b)^t \equiv r^t \equiv g^{H(m)}(g^a)^{H(r)} \equiv g^{H(m)+aH(g^b \mod q)} \mod q$$
und somit (die Gruppenordnung von $\mathbb{Z}_q^\times$ ist $q-1$ und g ist ein Erzeuger von $\mathbb{Z}_q^\times$)
$$bt \equiv H(m) + aH(g^b \mod q) \mod q - 1.$$
Wir erhalten $t \equiv b^{-1}(H(m) + aH(g^b \mod q)) \mod q - 1$. Das Tupel
$$(r,t) = (g^b \mod q, b^{-1}(H(m) + aH(g^b \mod q)) \mod q - 1)$$
ist somit eine mögliche Unterschrift von $m \in \mathscr{M}$ (unter der Wahl von $b \in \{2, \dots, q-2\}$), durchgeführt mit dem Signierschlüssel $s = (g,a) \in \mathscr{K}_S$. □

Bemerkung 4.23 (Sicherheitseigenschaften der ElGamal-Signatur)

1. Bei Nutzung der ElGamal-Signatur ist unter der Diffie-Hellman-Annahme weder ein Total Break noch universelle Fälschung bei einem Key-Only-Angriff in Polynomialzeit möglich. Dies kann analog zur Argumentation im Fall der RSA-Signatur gezeigt werden.
2. Es gibt Varianten der ElGamal-Signatur, die unter der Logarithmusannahme sicher gegen existentielle Fälschung unter einem Chosen-Message-Angriff sind.
3. Der *Digital Signature Algorithm (DSA)* funktioniert ähnlich der ElGamal-Signatur. Er wurde 1991 vom US-amerikanischen National Institute of Standards and Technology (NIST) vorgeschlagen und später zum Standard erklärt. ◁

4.2. Secret-Sharing

Es ist oft nützlich, private Schlüssel von Public-Key-Systemen rekonstruieren zu können, auch wenn der Besitzer sie verliert. Gleichzeitig wird erwartet, dass nicht ein Einzelner die Möglichkeit hat, private Schlüssel zu rekonstruieren, sondern dass mehrere Parteien, genannt Geheimnisträger, bei der Rekonstruktion zusammenarbeiten müssen. Dieses Ziel kann mit *Secret-Sharing-Protokollen* erreicht werden.

Definition 4.24 Seien $n, d \in \mathbb{N}$. Ein (n, d)*-Secret-Sharing-Protokoll* ist ein Verfahren, bei dem n Geheimnisträger (GT) Teilinformationen über ein Geheimnis erhalten, so dass

1. jeder Zusammenschluss von mindestens d Geheimnisträgern das Geheimnis rekonstruieren kann und
2. jeder Zusammenschluss von weniger als d Geheimnisträger keine relevante Information über das Geheimnis erhält. ◁

Wir beschreiben das *Shamir-Secret-Sharing-Protokoll*, das auf Eigenschaften von Polynomen basiert.

Definition 4.25 (Polynome) Sei K ein Körper. Ein *Polynom in einer Variablen über* K ist ein Ausdruck der Form

$$a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0$$

mit *Koeffizienten* $a_n, \ldots, a_0 \in K$, $a_n \neq 0$ und $n \in \mathbb{N}$. Wir fassen dabei t als *formale Variable* auf. (Natürlich könnte man hier auch einen anderen Buchstaben als formale Variable wählen, in diesem Skript werden wir die formale Variable eines Polynoms jedoch immer mit t bezeichnen.)

Die Zahl n, also der höchste in f auftretende Exponent von t wird der *Grad von* f genannt und als $\deg f$ geschrieben (die Bezeichnung kommt vom englischen Wort „degree“). Dann heißt $a_n \in K$ Leitkoeffizient von f.

Wir definieren $K[t]$ als die Menge aller Polynome über K. ◁

Definition 4.26 (Polynomfunktion) Ist $f = a_0 + a_1 t + \ldots + a_n t^n$ ein Polynom über dem Körper K und ist $x \in K$, so heißt

$$f(x) := a_0 + a_1 x + \ldots + a_n x^n \quad \in K$$

der *Wert von* f *in* x. Die zugehörige Funktion $K \to K$, $x \mapsto f(x)$, wird eine *Polynomfunktion* genannt. ◁

Beispiel 4.27 Jedes Polynom $f \in K[t]$ bestimmt eine Polynomfunktion $x \mapsto f(x)$. Zwei verschiedene Polynome können dabei dieselbe Polynomfunktion bestimmen:

Das Polynom $f = t^2 + t \in \mathbb{Z}_2[x]$ hat in jedem Element von $\mathbb{Z}_2$ den Wert Null: Es gilt $f([0]) = [0]^2 + [0] = [0]$ und $f([1]) = [1]^2 + [1] = [0]$. Während die beiden *Polynome*

$$t^2 + t \in \mathbb{Z}_2[t] \quad \text{und} \quad [0] \in \mathbb{Z}_2[t]$$

verschieden sind, sind somit die beiden zugehörigen *Polynomfunktionen*

$$t \mapsto t^2 + t \quad \text{und } t \mapsto [0]$$

gleich. Man beachte, dass man deshalb einer Polynomfunktion in der Regel keinen eindeutigen Grad zuweisen kann. Polynome verhalten sich hier also besser als Polynomfunktionen.

Man kann zeigen, dass Polynome und Polynomfunktionen über Körpern mit unendlich vielen Elementen (also z. B. $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$) übereinstimmen. In diesem (wichtigen) Fall macht es also keinen Unterschied, ob wir an Polynomfunktionen oder die formalen Ausdrücke aus Definition 4.25 denken. ◁

Das Shamir-Secret-Sharing-Protokoll basiert auf folgendem Lemma, das wir ohne Beweis zitieren.

Lemma 4.28 *Seien $q \in \mathbb{P}$ und $l \in \mathbb{N}$. Seien außerdem $x_i, y_i \in \mathbb{Z}_q$, $1 \leq i \leq l$ so gewählt, dass die x_i paarweise verschieden sind. Für $l \leq d \in \mathbb{N}$ gibt es dann genau q^{d-l} Polynome $f \in \mathbb{Z}_q[t]$ vom Grad höchstens $d-1$, die*

$$f(x_i) = y_i \quad \text{für alle } 1 \leq i \leq l$$

erfüllen, d. h. die alle Punkte (x_i, y_i) interpolieren. ◁

Konstruktion 4.29 (Shamir-Secret-Sharing-Protokoll) Alice möchte ein Geheimnis $s \in \mathbb{Z}_q$ so auf n Geheimnisträger verteilen, dass mindestens $d \leq n$ Geheimnisträger notwendig sind, um es zu rekonstruieren, und so dass ein Zusammenschluss von weniger als d Geheimnisträger keine Information über das Geheimnis erhält. Sie nutzt das Shamir-Secret-Sharing-Protokoll und geht wie folgt vor:

1. Alice veröffentlicht paarweise und von Null verschiedene Elemente $x_1, \dots, x_n \in \mathbb{Z}_q$ und legt öffentlich eine Zahl $d \in \mathbb{N}$ fest. Dadurch initialisiert sie das Protokoll.
2. Alice wählt zufällig gemäß der Gleichverteilung geheime Elemente $a_j \in \mathbb{Z}_q$, $1 \leq j \leq d-1$, und konstruiert daraus das Polynom

   $$f = s + \sum_{j=1}^{d-1} a_j t^j \in \mathbb{Z}_q[t]$$

   vom Grad höchstens $d-1$.
3. Alice berechnet $y_i = f(x_i)$, $1 \leq i \leq n$.
4. Alice übermittelt y_i an den i-ten Geheimnisträger, $1 \leq i \leq n$.

Das Geheimnis ist durch den konstanten Term $s = f([0]) \in \mathbb{Z}_q$ des Polynoms $f \in \mathbb{Z}_q[t]$ gegeben.

Beispiel 4.30 Die Anzahl der Geheimnisträger sei $n = 5$ und wir setzen $d = 3$. Alice wählt die Primzahl $q = 17$, die Elemente $x_i = [i] \in \mathbb{Z}_{17}$, $1 \leq i \leq n$, und veröffentlicht diese Informationen. Dadurch ist das Secret-Sharing-Protokoll initialisiert.

Alice möchte das Geheimnis $s = [3] \in \mathbb{Z}_{17}$ hüten. Sie wählt die geheimen Koeffizienten $a_i = [i + 13] \in \mathbb{Z}_{17}$ für $i \in \{1, 2\}$ und erhält dadurch das Polynom

$$f = [15]t^2 + [14]t + [3] \quad \in \mathbb{Z}_{17}[t].$$

Die fünf Geheimnisteile sind

1. $y_1 = f([1]) = [15]$,
2. $y_2 = f([2]) = [6]$,
3. $y_3 = f([3]) = [10]$,
4. $y_4 = f([4]) = [10]$,
5. $y_5 = f([5]) = [6]$.

Diese Geheimnisteile $y_1, \dots, y_5$ verteilt sie auf $n = 5$ verschiedene Geheimnisträger. Alice' Geheimnis ist $f([0]) = [3] \in \mathbb{Z}_{17}$. ◁

Konstruktion 4.31 (Geheimnisrekonstruktion) Alice hat das Polynom $f \in \mathbb{Z}_q[t]$ vom Grad maximal $d - 1 \in \mathbb{N}$ und die Stützstellen $x_1, \dots, x_n$ gewählt. Die Zahlen $q, d \in \mathbb{N}$ und $x_1, \dots, x_n \in \mathbb{Z}_q$ sind öffentlich zugänglich. Das Geheimnis $s = f([0])$ kann wie folgt durch die Zusammenarbeit von $d \leq n$ Geheimnisträgern rekonstruiert werden:

Wir nummerieren die Geheimnisträger so um, dass die Geheimnisteile $y_i = f(x_i)$ für $1 \leq i \leq d$ bekannt sind. Nun betrachten wir das Polynom

$$g = \sum_{k=1}^{d} y_k \prod_{j=1, j\neq k}^{d} \frac{x_j - t}{x_j - x_k} \quad \in \mathbb{Z}_q[t].$$

Das Polynom g erfüllt $g(x_i) = y_i$ für alle $1 \leq i \leq d$ und hat Grad $d - 1$.

Es gilt $q^{d-d} = q^0 = 1$. Gemäß Lemma 4.28 existiert somit genau ein Polynom $f' \in \mathbb{Z}_q[t]$ vom Grad höchstens $d - 1$, das $f'(x_i) = y_i$ für alle $1 \leq i \leq d$ erfüllt. Wir wissen, dass sowohl das von Alice gewählte geheime Polynom f als auch das oben definierte Polynom g diese Bedingungen erfüllen. Folglich muss $f = g$ gelten. Das Geheimnis s ist also durch den konstanten Term von g gegeben:

$$s = g([0]) = \sum_{k=1}^{d} y_k \prod_{j=1, j\neq k}^{d} \frac{x_j}{x_j - x_k} \quad \in \mathbb{Z}_q$$

Da die Stützstellen $x_i \in \mathbb{Z}_q$, $1 \leq i \leq n$, öffentlich sind, kann mit dieser Formel jeder Zusammenschluss aus d Geheimnisträgern das Geheimnis $s \in \mathbb{Z}_q$ rekonstruieren. ◁

Beispiel 4.32 Wir setzten das obige Beispiel fort. Die ersten $d = 3$ Geheimnisträger können aus $y_1 = [15]$, $y_2 = [6]$ und $y_3 = [10]$ Alice' Geheimnis s rekonstruieren:

$$\begin{aligned} s = f([0]) &= [15] \cdot \frac{[2]}{[2-1]} \cdot \frac{[3]}{[3-1]} + [6] \cdot \frac{[1]}{[1-2]} \cdot \frac{[3]}{[3-2]} + [10] \cdot \frac{[1]}{[1-3]} \cdot \frac{[2]}{[2-3]} \\ &= [3] \quad \in \mathbb{Z}_{17} \end{aligned}$$

◁

Wir zeigen nun, dass ein Zusammenschluss von $d > m \in \mathbb{N}$ Geheimnisträgern keine relevante Information über das Geheimnis erhält. Dazu nehmen wir an, dass die Geheimnisteile $y_1, \dots, y_m$ bekannt sind. Außerdem ist bekannt, dass das Geheimnis durch den konstanten Term $s = f([0])$ eines Polynoms $f \in \mathbb{Z}_q[t]$ vom Grad höchstens $d - 1$ gegeben ist:

Sei $s' \in \mathbb{Z}_q$ beliebig. Wir wählen $x_{m+1} = [0] \in \mathbb{Z}_q$ und $y_{m+1} = s' \in \mathbb{Z}_q$. Dann folgt mit Lemma 4.28 (für $l = m+1$), dass genau $0 < q^{d-(m+1)}$ Polynome $f' \in \mathbb{Z}_q[t]$ vom Grad höchstens $d-1$ existieren, die $f'(x_i) = y_i$ für alle $1 \leq i \leq m+1$ erfüllen.

Insbesondere gibt es für jeden möglichen Wert des Geheimnisses $s' \in \mathbb{Z}_q$ gleich viele Polynome mit dieser Eigenschaft. Da die Koeffizienten $a_i \in \mathbb{Z}_q$, $1 \leq i \leq d-1$, des geheimen Polynoms

$$f = s + \sum_{i=1}^{d-1} a_i t^i \quad \in \mathbb{Z}_q[t]$$

von Alice zufällig gemäß der Gleichverteilung gewählt wurden, sind alle möglichen Werte $s' \in \mathbb{Z}_q$ für den konstanten Term $f([0])$ des Polynoms f (unter den gegebenen Daten) gleich wahrscheinlich. Es folgt, dass ein Zusammenschluss von $d > m$ Geheimnisträgern aus der Kenntnis von $y_1, \ldots, y_m$ keinerlei Information über das Geheimnis $s = f([0]) \in \mathbb{Z}_p$ erhält.

Teil II.

Informations- und Codierungstheorie

Empfohlene Literatur

David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. URL: http://www.inference.phy.cam.ac.uk/itprnn/book.html (sehr ausführlich und anschaulich; mathematisch teilweise unsauber)

Robert B. Ash. *Information Theory*. Dover, 1990 (Standardwerk)

Wolfgang Willems. *Codierungstheorie und Kryptographie*. Birkhäuser, 2008. URL: http://link.springer.com/book/10.1007/978-3-7643-8612-2

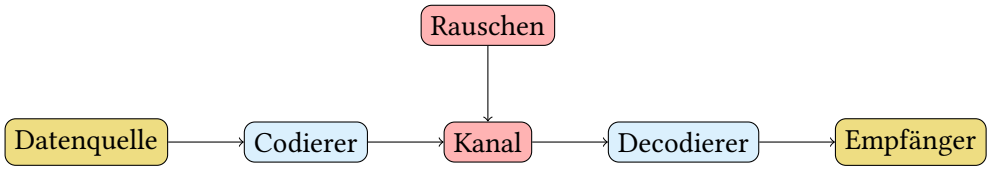
Götz Kersting und Anton Wakolbinger. *Elementare Stochastik*. Birkhäuser Verlag AG, 2008. URL: http://link.springer.com/book/10.1007/978-3-7643-8431-9 (Sehr gutes deutsches Buch zur Stochastik mit einem kurzen Kapitel zur Informationstheorie)

Einführung

Die heutige Zeit ist durch die Allgegenwart *digitaler Information* und deren Verbreitung bzw. Verarbeitung geprägt. Zur Informationsübertragung werden verschiedene Medien wie Telefon- und Netzwerkkabel, die Luft (Mobilfunk, Radio, DVB-T), der Weltraum (Satellitenkommunikation), aber auch DVDs oder USB-Sticks genutzt.

All diesen *Kommunikationskanälen* (den Ausdruck werden wir später präzise definieren) ist gemein, dass sie von unausweichlichem Rauschen bei der Datenübertragung betroffen sind; sei es aufgrund von Umwelteinflüssen auf dem Übertragungskanal (elektromagnetische Störungen, Wettereffekte, Kratzer, ...) oder technische Unzulänglichkeiten der beteiligten Elektronik (Qualität der sendenden / empfangenden Antennen, Schaltkreise zur Signalverarbeitung, etc.). Die meisten Anwendungen setzen jedoch eine praktisch fehlerfreie Übertragung voraus; lädt man etwa ein Computerprogramm herunter, kann schon ein einziges fehlerhaftes Bit dafür sorgen, dass das Programm nicht funktioniert.

Die *Informationstheorie* beschäftigt sich daher mit der Analyse von Kommunikationssystemen, die man schematisch wie folgt darstellen kann:



Die Datenquelle besteht aus einer Person oder einer Maschine, welche die zu übertragende Information produziert. Der Codierer hat die Aufgabe, diese Nachricht in eine Form zu übersetzen, die zur Übertragung auf dem Kanal geeignet ist. Dies kann je nach Anwendung z. B. eine Folge von Bits oder auch ein elektromagnetisches Signal sein. Außerdem soll die codierte Nachricht robust

gegen die durch das Kanalrauschen eintretenden Übertragungsfehler sein; der Codierer muss also irgendeine Form von *Redundanz* einbauen, so dass der *Decodierer* Fehler erkennen oder sogar korrigieren kann. Allerdings reduziert diese Redundanz die nutzbare Übertragungsrate des Kanals, so dass wir es mit zwei entgegengesetzten Zielen – niedrige Fehlerrate, aber hohe Übertragungsrate – zu tun haben.

Beispiel 4.33 Eine Datenquelle produziere fortlaufend Bits, also Buchstaben aus dem Alphabet $\mathrm{GF}(2) = \{0, 1\}$, mit der Rate 1 bit/s. Die Werte 0 und 1 sollen dabei mit gleicher Wahrscheinlichkeit auftreten und die einzelnen Bits unabhängig voneinander verteilt sein, so dass die Quelle dem wiederholten Werfen einer perfekten Münze entspricht. Die Bits werden über einen ebenfalls binären Kanal übertragen, der höchstens einmal pro Sekunde in Anspruch genommen werden kann, der jedoch mit Wahrscheinlichkeit $\varepsilon = \frac{1}{4}$ ein Bit falsch überträgt, also aus der 0 eine 1 macht oder umgekehrt (auch der Kanal soll aufeinanderfolgende Übertragungen unabhängig voneinander beeinflussen).

Ist nun die Fehlerrate von $\varepsilon = \frac{1}{4}$ für unsere Zwecke zu hoch (in der Praxis sind oft Garantien in der Größenordnung $\varepsilon < 10^{-15}$ gefordert), müssen wir die Zuverlässigkeit des Systems erhöhen. Die einfachste Möglichkeit ist, jedes von der Quelle emittierte Bit mehrmals, z. B. dreimal, zu senden. Der Codierer besteht dann aus der Funktion $E\colon \mathrm{GF}(2) \to \mathrm{GF}(2)^3$, $E(a) = aaa$. Der Decodierer führt für jedes empfangene Tripel eine „Mehrheitsentscheidung“ durch, decodiert es also genau dann als 1, wenn es mindestens zwei Einser enthält:

$$\begin{aligned} &D\colon \mathrm{GF}(2)^3 \to \mathrm{GF}(2) \\ &D(a_1a_2a_3) = \begin{cases} 0, & \text{falls } |\{i \in \{1,2,3\} \colon a_i = 0\}| \geq 2 \\ 1 & \text{sonst.} \end{cases} \end{aligned}$$

Solange der Kanal höchstens eines der drei Bits verfälscht, kann der Decodierer so das gesendete Bit korrekt rekonstruieren; andernfalls (2 oder 3 Fehler) liegt er falsch.

Die *Bit-Fehlerwahrscheinlichkeit*, also die Wahrscheinlichkeit, dass ein gesendetes Nutzdaten-Bit beim Empfänger (nach dem Decodieren) falsch ankommt, liegt damit bei

$$P(\text{2 oder 3 Fehler}) = \binom{3}{2}\left(\frac{1}{4}\right)^2\frac{3}{4} + \binom{3}{3}\left(\frac{1}{4}\right)^3\left(\frac{3}{4}\right)^0 = \frac{10}{64} \approx 0{,}16 < \frac{1}{4}.$$

Wir haben also die Fehlerwahrscheinlichkeit reduziert; der Preis dafür ist allerdings, das wir die Quelle auf 1/3 bit/s drosseln müssen, da der Kanal ja nur 1 bit/s annimmt – anders gesagt, pro Kanal-Bit wird im Schnitt nur noch 1/3 Nutzdaten-Bit übertragen. Das Verhältnis von Quell-Bitrate zu Kanal-Bitrate wird *Codierungsrate* oder einfach *Rate* des Systems genannt. ◁

Aufgabe 4.34 Verallgemeinern Sie das Verfahren aus Beispiel 4.33 auf beliebige Fehlerwahrscheinlichkeit des Kanals $\varepsilon < \frac{1}{2}$ und n-fache Bitwiederholung für ungerade n. Zeigen Sie: Mit wachsendem n geht zwar die Wahrscheinlichkeit eines Decodierfehlers gegen 0, die Rate allerdings auch. ◁

Die Informationstheorie geht der Frage nach, was hier überhaupt theoretisch erreichbar ist: Wie klein muss die Rate werden, um eine bestimmte, noch akzeptable Fehlerwahrscheinlichkeit erreichen zu können? Das wichtigste Resultat besagt, dass es (anders als beim Verfahren in Aufgabe 4.34) zum Erreichen einer beliebig kleinen Fehlerwahrscheinlichkeit *nicht* notwendig ist, die Rate bis auf 0 zu reduzieren: solange die Rate unterhalb der *Kapazität* des Kanals liegt (die wir später noch definieren werden), können wir die Fehlerwahrscheinlichkeit durch geeignete Codierungsverfahren beliebig reduzieren, *ohne die Rate weiter zu senken.* Im ersten Kapitel dieser Vorlesungshälfte (Kapitel 5) werden wir dieses Resultat formalisieren und beweisen. Da der Beweis leider nicht konstruktiv ist, geht es danach in der *Codierungstheorie* (Kapitel 6) um konkrete Codierungsverfahren, die diese Kapazität zu erreichen versuchen.

Zunächst werden wir allerdings von einem fehlerfreien Kanal ausgehen und untersuchen, wie wir seine Übertragungsrate durch Codierung der Nachrichten möglichst effizient ausnutzen können. Denken wir beim Kanal an ein digitales Speichermedium, entspricht das der äquivalenten Frage, wie man die Daten aus der Quelle mit möglichst geringem Speicherverbrauch codieren kann (*Datenkompression*). Dabei begegnen wir dem Begriff der *Entropie* oder *mittleren Ungewissheit* einer Nachrichtenquelle und werden sehen, dass die Entropie mit der optimal erreichbaren Kompression zusammenhängt.

5. Informationstheorie

5.1. Kommunikation über fehlerfreie Kanäle / Datenkompression

Wir gehen in diesem Kapitel von einer fehlerfreien Übertragung ohne Rauschen aus:

Der Kanal gebe beim Empfänger also immer genau das aus, was beim Sender eingegeben wurde. Der Einfachheit halber beschränken wir uns auf einen *binären* Kanal, der Bits (Buchstaben von $\mathrm{GF}(2)$) überträgt. Da es keine Übertragungsfehler zu berücksichtigen gibt, bleibt von den in der Einführung formulierten Zielen nur noch die möglichst hohe Übertragungsrate übrig. Wir gehen also folgender Frage nach:

Frage 5.1 Wie kann ein Codierer die Daten aus der Datenquelle möglichst sparsam zu binären Nachrichten codieren, wobei gewährleistet bleiben muss, dass der Decodierer die Daten wieder eindeutig rekonstruieren kann?

Stellt man sich unter dem (abstrakten) Kanal ein Speichergerät vor, haben wir die äquivalente Frage, wie man gegebene Daten mit möglichst geringem Speicherverbrauch abspeichern kann.◁

Dazu formalisieren wir zunächst den Begriff der Quelle, die nichts weiter ist als eine Zufallsvariable.

Definition 5.2 (Quelle) Eine *(Daten-)quelle* (A, X) besteht aus einem Alphabet $A = \{a_1, \dots, a_s\}$, $s = |A|$ und einer Zufallsvariable X mit Zielbereich A, welche die Wahrscheinlichkeit eines einzelnen Buchstabens

$$p_i := P_X(a_i) = P(X = a_i)$$

der Quelle angibt. Die Buchstaben werden auch *Symbole* oder *Nachrichten* genannt, die Zufallsvariable X nennen wir einen *zufälligen Buchstaben.*

Wir nehmen durchweg an, dass die Quelle beliebig viele Buchstaben generieren kann, die alle unabhängig voneinander nach P_X verteilt sind. Die N-dimensionale Zufallsvariable X^N ist dann ein *zufälliges N-Wort* aus der Quelle, mit $P_{X^N}(x_1 \cdots x_N) = \prod_{i=1}^{N} P_X(x_i)$. ◁

Bemerkung 5.3 In der Schule oder auch in der Stochastik-Vorlesungen werden Wahrscheinlichkeiten oft benutzt, um die relative Häufigkeit eines beliebig oft wiederholbaren Experiments (beispielsweise des Werfens einer Münze) zu beschreiben. In unserem Kontext ist es hilfreich, sich die Wahrscheinlichkeit viel allgemeiner als ein Maß dafür vorzustellen, wie sicher bzw. unsicher man sich darin ist, dass ein bestimmtes Ereignis eintritt. Dieses ist unweigerlich subjektiv: schickt etwa eine Person A eine SMS mit dem Endstand seines Fußballspiels an B, so ist deren Inhalt natürlich nicht zufällig, sondern eindeutig durch das bereits stattgefundene Spiel determiniert. Aus Sicht von B, der das Spiel nicht gesehen hat, kann der Spielstand vor Eintreffen der Nachricht aber durchaus durch eine Zufallsvariable modelliert werden, in deren Verteilung z. B. seine Einschätzung der Spielstärken beider Mannschaften einfließen können.◁

5.1.1. Kompaktkurs Stochastik II

Wir wiederholen einige weitere stochastische Grundlagen.

Definition 5.4 (Erwartungswert und Varianz) Der *Erwartungswert* einer reellen Zufallsvariable X mit Alphabet $A = \{a_1, \dots, a_s\}$ ist definiert als

$$\mu_X = \mathbb{E}[X] = \sum_{i=1}^{s} a_i P(X = a_i) = \sum_{i=1}^{s} a_i p_i.$$

Als ihre *Varianz* bezeichnet man die erwartete quadratische Abweichung vom Mittelwert:

$$\sigma_X^2 = \mathbb{E}[(X - \mathbb{E}[X])^2]$$ ◁

Satz 5.5 (Schwaches Gesetz der großen Zahlen) *Seien $X_1, \dots, X_N$ unabhängige Zufallsvariablen, alle mit dem gleichen Erwartungswert $\mu = \mu_{X_i}$ und Varianz $\sigma^2 = \sigma_{X_i}^2$. Sei $\bar{X}_N = \frac{1}{N}\sum_{n=1}^{N} X_n$ der Stichproben-Mittelwert von $X_1, \dots, X_N$. Dann gilt:*

$$P\left((\bar{X}_N - \mu)^2 \geq \alpha\right) \leq \frac{\sigma^2}{\alpha N}$$

für alle $\alpha > 0$: $\bar{X}_N$ nähert sich μ mit wachsendem N immer mehr an. ◁

Bemerkung 5.6 Sei Ω ein Wahrscheinlichkeitsraum und $X\colon \Omega \to A$ eine Zufallsvariable. Ist $h\colon A \to A'$ eine Abbildung von A in eine Menge A', wird die Verkettung $h \circ X$ zu einer neuen Zufallsvariablen mit Zielbereich A', die wir mit $h(X)$ notieren (Abbildung 5.1). Ihre Verteilung ist gegeben durch

$$P_{h(X)}(S') = P(h(X) \in S') = P(\omega \in \Omega\colon h(X(\omega)) \in S') = \sum_{\substack{a \in A \\ h(a) \in S'}} P_X(a)$$

für $S' \subseteq A'$. ◁

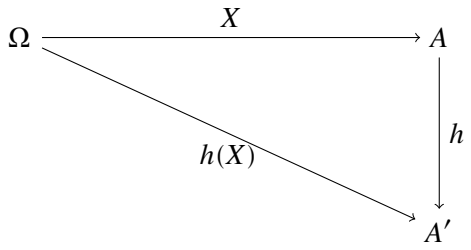


Abbildung 5.1.: Verkettung der Zufallsvariable X mit einer Abbildung h zur neuen Zufallsvariable $h(X)$ (Bemerkung 5.6).

5.1.2. Symbolcodes

Zum Codieren verwenden wir *Symbolcodes*, die jedem Buchstaben des Quellalphabets ein bestimmtes binäres Wort zuweisen.

Definition 5.7 (Symbolcode) Sei (A, X) eine Quelle. Ein (binärer) *Symbolcode* für A ist eine injektive Abbildung

$$E\colon A \to \mathrm{GF}(2)^* \setminus \{\varepsilon\}.$$

Sie ordnet jedem Buchstaben $a_i \in A$ ein nichtleeres *Codewort* $E(a_i)$ über $\mathrm{GF}(2)$ zu, dessen Länge wir mit $l_i := l(E(a_i))$ bezeichnen. Der Erwartungswert

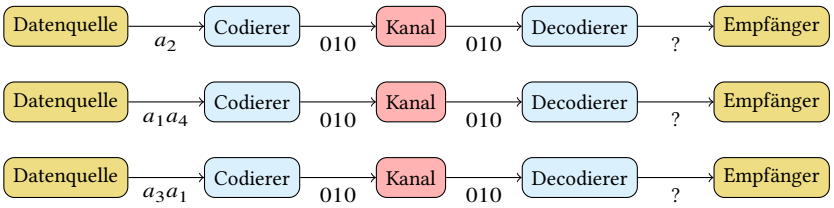
$$l_E := \mathbb{E}[l(E(X))] = \sum_{a\in A} P_X(a) l(E(a)) = \sum_{i=1}^{s} l_i p_i$$

wird *mittlere Wortlänge* von E genannt. ◁

Es ist klar, dass wir, um Frage 5.1 zu beantworten, einen Symbolcode E mit möglichst kleiner mittleren Wortlänge l_E finden müssen. Dabei muss allerdings sichergestellt sein, dass die beim Empfänger ankommenden (binären) Nachrichten auch wieder decodiert werden können. Bei einzelnen Buchstaben ist dies kein Problem, da E nach Voraussetzung injektiv ist. Wird jedoch eine Folge von Buchstaben $w_1 \cdots w_n \in A^*$ zu $E(w_1) \cdots E(w_n) \in \mathrm{GF}(2)^*$ codiert, treten Probleme auf. Sei zum Beispiel $A = \{a_1, a_2, a_3, a_4\}$ und der Code gegeben durch

$$E(a_1) = 0, \quad E(a_2) = 010, \quad E(a_3) = 01, \quad E(a_4) = 10$$

Die Bitfolge 010 könnte durch Codierung von a_2, a_1a_4 oder a_3a_1 entstanden sein; kein Decodierer könnte dies unterscheiden!



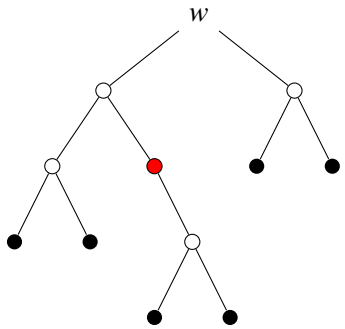


Abbildung 5.2.: Beispiel für einen Binärbaum. Die Wurzel ist mit w markiert, innere Knoten durch nicht ausgefüllte, Blätter durch ausgefüllte Punkte. Der Baum ist nicht voll, da der markierte Knoten nicht nach links verzweigt.

Wir beschränken uns daher auf *Präfixcodes*, bei denen solche Mehrdeutigkeiten ausgeschlossen sind.

Definition 5.8 (Präfixcode) Ein Symbolcode E heißt *Präfixcode*, wenn kein Codewort Präfix (Anfangsstück) eines anderen ist, wenn also für $a_i \neq a_j \in A$ kein $w \in \mathrm{GF}(2)^*$ mit $E(a_i)w = E(a_j)$ existiert. ◁

Beispiel 5.9 Bekanntestes Beispiel für Präfixcodes bilden die Telefonnummern: keine Telefonnummer ist Präfix einer anderen Telefonnummer, da es sonst beim Wählprozess zwischendurch bei einem anderen Anschluss klingeln würde. ◁

Lemma 5.10 *Ist $w = w_1 \cdots w_k \in A^*$ und E ein Präfixcode, so kann $E(w) = E(w_1) \cdots E(w_k)$ eindeutig decodiert werden: es gibt kein $w' = w'_1 \cdots w'_{k'} \neq w$ mit $E(w_1) \cdots E(w_k) = E(w'_1) \cdots E(w'_{k'})$.* ◁

Präfixcodes lassen sich elegant durch *Binärbäume* darstellen: Ein Binärbaum besitzt einen Knoten w, genannt *Wurzel*, den wir uns als ganz „oben“ vorstellen. Von w und jedem anderen Knoten zweigen nach unten höchstens zwei Kanten ab, wobei jede Kante entweder nach links oder rechts verläuft. Ein Knoten, aus dem keine Kante nach unten herausführt, heißt *Blatt*, sonst *innerer Knoten*. Wenn aus jedem inneren Knoten zwei Kanten nach unten führen heißt der Baum *voll* (siehe Abbildung 5.2).

Für einen Präfixcode E sind im entsprechenden Binärbaum die Buchstaben von A den Blättern des Baums zugewiesen, und zwar so dass die Codierung $E(a_i) \in \mathrm{GF}(2)^*$ dem (eindeutigen) Kantenzug von der Wurzel w zum Blatt a_i entspricht: dabei schreiben wir für jede Kante nach links unten eine 0, nach rechts unten eine 1. Die Präfixeigenschaft des Codes garantiert, dass Buchstaben immer Blätter sind, dass wir auf dem Weg von der Wurzel zu einem Buchstaben a_i also nie an einem anderen $a_j \neq a_i$ vorbeikommen. Die *Tiefe* des Baums, definiert als die maximale Länge eines Kantenzugs von w zu einem Blatt, entspricht dann gerade der maximalen Codewortlänge von E. Abbildung 5.3 zeigt beispielhaft einen Codebaum für ein Alphabet mit 5 Buchstaben.

$a_i \in A$	$E(a_i)$
a_1	10
a_2	001
a_3	01
a_4	000

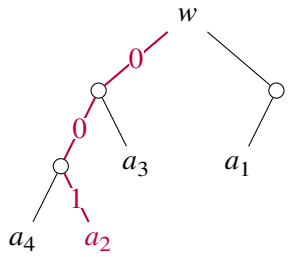


Abbildung 5.3.: Baumdarstellung eines Präfixcodes mit $|A| = 4$ und maximaler Codewortlänge $l_2 = l_4 = 3$. Markiert ist die Codierung von a_2.

Um zur Beantwortung von Frage 5.1 die mittlere Wortlänge zu minimieren, müssen wir eine optimale Konfiguration von Wortlängen für eine gegebene Quelle (A, X) finden. Der folgende Satz gibt vor, welche Längen für ein gegebenes Alphabet überhaupt möglich sind.

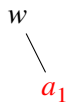
Satz 5.11 (Kraft-Ungleichung) *Sei $A = \{a_1, \dots, a_s\}$. Ein Präfixcode mit den Wortlängen $l_1, \dots, l_s$ existiert genau dann, wenn $\sum_{i=1}^{s} 2^{-l_i} \leq 1$ gilt. Dabei gilt Gleichheit genau dann, wenn der dem Präfixcode entsprechende Binärbaum voll ist.* ◁

Beweis. „⇒“: Sei E ein Präfixcode. Wir stellen uns folgendes „Spiel“ vor: von der Wurzel des E entsprechenden Binärbaums ausgehend werfen wir an jeder Abzweigung eine faire Münze und wählen gemäß dem Ergebnis eine der beiden Kanten. Wir stoppen, wenn wir entweder in einem Codewort oder einem „toten Ast“ des Baums (in Abbildung 5.3 entspricht das der Folge 11) landen. Dann erreichen wir $a_i \in A$ mit Wahrscheinlichkeit $\left(\frac{1}{2}\right)^i = 2^{-l_i}$. Wir erhalten also einen Wahrscheinlichkeitsraum Ω, dessen Elementarereignisse die Buchstaben $a_i \in A$ sowie das Ereignis „toter Ast“ sind. Wir erhalten so $1 = P(\Omega) = \sum_{i=1}^{s} P(a_i) + P(\text{toter Ast}) \geq \sum_{i=1}^{s} 2^{-l_i}$. Strikte Ungleichung gilt genau für $P(\text{toter Ast}) > 0$, wenn also der Baum nicht voll ist.
(*Achtung:* die Wahrscheinlichkeiten hier dürfen nicht mit der Wahrscheinlichkeit verwechselt werden, dass die Quelle einen Buchstaben a_i erzeugt; die Wahrscheinlichkeitsverteilung der Quelle wird hier noch gar nicht berücksichtigt)

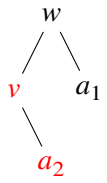
„⇐“: Seien nun Wortlängen $l_1, \dots, l_s$ mit $\sum_{i=1}^{s} 2^{-l_i} \leq 1$ gegeben; wir nehmen oBdA an, dass $l_1 \leq \dots \leq l_s$ gilt. Wir zeigen per Induktion über k, $1 \leq k \leq s$: *Es existiert ein Präfixcode E_k über dem Teilalphabet $A_k := \{a_1, \dots, a_k\}$ mit Längen $l_1, \dots, l_k$. Der Binärbaum T_k zu E_k ist darüberhinaus für $k < s$ nicht voll.*

Induktionsanfang ($k = 1$): Definiere $E_1(a_1) = \underbrace{1 \cdots 1}_{l_1 \text{ mal}}$; der Baum T_1 ist nicht voll, da die Wurzel nach links frei bleibt.

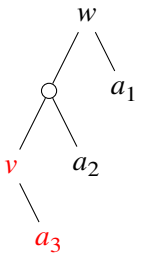
Induktionsschluss ($k - 1 \to k$): Setze $E_k(a_i) = E_{k-1}(a_i)$ für $1 \leq i \leq k - 1$ und lege $E_k(a_k)$ mit Länge l_k wie folgt fest: nach Voraussetzung ist der zu E_{k-1} gehörende Baum T_{k-1} mit Tiefe l_{k-1} nicht voll, wir können also mindestens noch einen Knoten v der Tiefe l_{k-1} einbauen. Ist $l_k = l_{k-1}$, definieren wir $E(a_k)$ gemäß v; ist $l_k > l_{k-1}$, zweigen wir von v so oft nach rechts unten ab, bis die gewünschte Tiefe l_k erreicht ist.



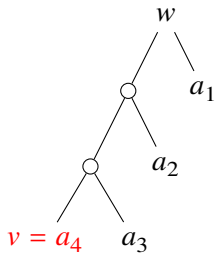
(a) Codieren von a_1 mit $l_1 = 1$ zu $E(a_1) = 1$.



(b) Codieren von a_2 mit $l_2 = 2$. Da $l_2 > l_1$, fügen wir den inneren Knoten v ein und zweigen dann nach rechts ($E(a_2) = 01$).



(c) Codieren von a_3 mit $l_3 = 3$, analog zu a_2.



(d) Codieren von a_4 mit $l_4 = 3$. Da $l_4 = l_3$, wird die Tiefe des Baums nicht erhöht, sondern a_4 direkt als neues Blatt neben a_3 codiert.

Abbildung 5.4.: Konstruktion eines Codes nach dem Beweis der Kraft-Ungleichung (Beispiel 5.12).

Für $k < s$ gilt $\sum_{i=1}^{k} 2^{-l_i} < \sum_{i=1}^{s} 2^{-l_i} \leq 1$, also ist $\sum_{i=1}^{k} 2^{-l_i} < 1$ strikt erfüllt, was mit dem gerade bewiesenen ersten Teil des Satzes impliziert, dass T_k nicht voll ist. □

Beispiel 5.12 Sei $A = \{a_1, \dots, a_4\}$ und $l_1 = 1$, $l_2 = 2$, $l_3 = l_4 = 3$. Es gilt $\sum_{i=1}^{4} 2^{-l_i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1$; die Kraft-Ungleichung ist also erfüllt. Nach Satz 5.11 existiert ein Präfixcode mit vollem Binärbaum, der die geforderten Längen hat. Die Anwendung der Methode aus dem Beweis von Satz 5.11 ist in Abbildung 5.4 dargestellt.

5.1.3. Optimales Codieren zufälliger Buchstaben

In diesem Kapitel werden wir gute Symbolcodes konstruieren, die auf die Verteilung eines zufälligen Buchstabens X zugeschnitten sind, um die mittlere Wortlänge zu minimieren. Für diese werden wir eine untere Schranke herleiten und zeigen, dass unsere Codes diese Schranke (fast) erreichen.

Sehen wir uns zunächst den Spezialfall an, dass X gleichverteilt ist und außerdem $|A| = 2^L$ für ein $L \in \mathbb{N}$ gilt. Dann ist $p_i = 2^{-L}$ für alle $a_i \in A$, und es ist intuitiv am besten, alle Buchstaben zu Wörtern der gleichen Länge L zu codieren, entsprechend den Blättern eines vollständigen Binärbaums der Tiefe L. Die Bedingung $p_i = 2^{-L}$ heißt analog $L = -\log_2 p_i$. Die grundlegende

Idee von *Shannon-Codes* besteht darin, die Codierungslänge eines Buchstabens $a_i \in A$ auch für beliebige Verteilungen durch $-\log_2 p_i$ zu bestimmen.

Notation 5.13 Für den Rest der Vorlesung gilt $\log x := \log_2 x$, beim Logarithmus ist also immer die Basis 2 impliziert. Den natürlichen Logarithmus (Basis e) bezeichnen wir wie üblich mit $\ln x$. ◁

Definition 5.14 (Shannon-Code) Seien (A, X) eine Quelle mit $P_X(a_i) = p_i$ für $a_i \in A$. Ein Präfixcode E heißt *Shannon-Code für* (A, X), wenn $l_i = \lceil -\log p_i \rceil$ für alle $i \in \{1, \dots, s\}$ gilt, wenn also l_i die eindeutige natürliche Zahl mit

$$-\log p_i \leq l_i < -\log p_i + 1 \tag{5.1}$$

ist. ◁

Aus (5.1) folgt $2^{-l_i} \leq 2^{\log p_i} = p_i$, so dass $\sum_{i=1}^{s} 2^{-l_i} \leq \sum_{i=1}^{s} p_i = 1$ gilt; nach Satz 5.11 existiert also für jede Quelle (A, X) ein Shannon-Code.

Definition 5.15 (Entropie) Sei X eine Zufallsvariable mit Alphabet $A = \{a_1, \dots, a_s\}$ und $p_i = P_X(a_i)$. Dann wird

$$H(X) = -\sum_{i=1}^{s} p_i \cdot \log p_i = \mathbb{E}[-\log P_X(X)]$$

die *Entropie* von X genannt. Dabei (und im Folgenden) definieren wir $0 \log 0 := 0$. ◁

Bemerkung 5.16 Der Erwartungswert in Definition 5.15 ist über einer Zufallsvariable definiert, die wie in Bemerkung 5.6 durch Verkettung von X mit der Abbildung $h\colon A \to \mathbb{R}$, $a_i \mapsto -\log P_X(a_i)$ gebildet wird. h bildet also Elemente aus A auf etwas ab, was von deren Wahrscheinlichkeit abhängt! Dieses ungewöhnliche Konzept wird uns noch öfters begegnen.◁

Für einen Shannon-Code erhalten wir als Abschätzung der erwarteten Wortlänge

$$l_E = \sum_{i=1}^{s} \lceil -\log p_i \rceil p_i \geq -\sum_{i=1}^{s} p_i \log p_i = H(X).$$

Wie der folgende Satz zeigt, ist die Entropie sogar untere Schranke für *alle* Präfixcodes.

Satz 5.17 (Quellencodierungssatz für Symbolcodes (schwache Version)) *Für jeden Präfixcode E auf einer Quelle (A, X) gilt $l_E \geq H(X)$, für Shannon-Codes außerdem $l_E < H(X) + 1$.* ◁

Zum Beweis von Satz 5.17 benötigen wir zunächst zwei weitere Aussagen.

Lemma 5.18 *Es gilt $\log x \leq \frac{1}{\ln 2}(x - 1)$ für alle $x > 0$ mit Gleichheit nur für $x = 1$.* ◁

Beweis. Übungsaufgabe. □

Lemma 5.19 (Kullback-Leibler-Abstand) *Seien X und Y Zufallsvariablen auf dem gleichen Alphabet $A = \{a_1, \dots, a_s\}$ mit Verteilungen $P_X(a_i) = p_i$, $P_Y(a_i) = q_i$. Dann gilt:*

$$D_{\mathrm{KL}}(P_X \parallel P_Y) := \sum_{i=1}^{s} p_i \log \frac{p_i}{q_i} \geq 0 \tag{5.2}$$

oder äquivalent

$$H(X) = -\sum_{i=1}^{s} p_i \log p_i \leq -\sum_{i=1}^{s} p_i \log q_i \tag{5.3}$$

mit Gleichheit genau dann, wenn $p_i = q_i$ für alle i. $D_{\mathrm{KL}}(P_X \parallel P_Y)$ wird auch Kullback-Leibler-Abstand *oder* relative Entropie *von P_X bezüglich P_Y genannt.* ◁

Beweis.

$$\begin{aligned} D_{\mathrm{KL}}(P_X \parallel P_Y) = \sum_{i=1}^{s} p_i \log \frac{p_i}{q_i} &= -\sum_{i=1}^{s} p_i \log \frac{q_i}{p_i} \\ &\geq -\frac{1}{\ln 2} \sum_{i=1}^{s} p_i \left(\frac{q_i}{p_i} - 1 \right) = -\frac{1}{\ln 2} \left(\sum_{i=1}^{s} q_i - \sum_{i=1}^{s} p_i \right) = 0 \end{aligned}$$

wobei die Ungleichung aus Lemma 5.18 folgt. Mit der zweiten Aussage von Lemma 5.18 folgt, dass Gleichheit genau dann gilt, wenn alle $\frac{q_i}{p_i} = 1$ sind, also $p_i = q_i$ für $i = 1, \dots, s$ gilt. □

Beweis (von Satz 5.17). Sei ein beliebiger Präfixcode E über dem Alphabet $A = \{a_1, \dots, a_s\}$ mit Wortlängen $l_1, \dots, l_s$ gegeben. Zum Beweis der ersten Aussage schreiben wir diese zunächst um: Mit den Definitionen $H(X) = \sum_{i=1}^{s} p_i \log p_i$ und $l_E = \sum_{i=1}^{s} p_i l_i$ sowie der Umformung $l_i = -\log 2^{-l_i}$ erhalten wir die Äquivalenz

$$H(X) \leq l_E \Leftrightarrow -\sum_{i=1}^{s} p_i \log p_i \leq -\sum_{i=1}^{s} p_i \log 2^{-l_i}.$$

Dies sieht verdächtig nach (5.3) aus – das Problem ist nur, dass $(2^{-l_i})_{i=1}^{s}$ nicht zwingend eine Wahrscheinlichkeitsverteilung ist, also $S := \sum_i 2^{-l_i}$ kleiner als 1 sein könnte. Zur Reparatur dieses Problems definieren wir $q_i = \frac{2^{-l_i}}{S}$ und erhalten so $\sum_i q_i = 1$; also gilt nach (5.3)

$$H(X) \leq -\sum_{i=1}^{s} p_i \log q_i = -\sum_{i=1}^{s} p_i \log \frac{2^{-l_i}}{S} = -\sum_{i=1}^{s} p_i \log 2^{-l_i} + \sum_{i=1}^{s} p_i \log S = l_E + \log S \leq l_E$$

da $S \leq 1$ nach Satz 5.11, also $\log S \leq 0$.

Für einen Shannon-Code folgt mit der zweiten Ungleichung aus (5.1)

$$l_E = \sum_{i=1}^{s} p_i l_i < \sum_{i=1}^{s} p_i \cdot (-\log p_i + 1) = H(X) + 1,$$

womit die zweite Behauptung gezeigt ist. □

Nach Satz 5.17 benötigen wir mit Präfixcodes im Mittel mindestens $H(X)$ und mit Shannon-Codes höchstens $H(X) + 1$ Bits, um einen zufälligen Buchstaben X einer Quelle zu codieren. Man kann zeigen, dass die untere Schranke nicht nur für Präfixcodes, sondern für *alle* eindeutig decodierbaren Codes gilt. Somit beantworten Shannon-Codes Frage 5.1 bis auf eine verbleibende Lücke von bis zu einem Bit pro Buchstaben; diese werden wir schließen können, nachdem wir in Abschnitt 5.1.4 die Entropie genauer untersucht haben.

Bemerkung 5.20 In unserem Beispiel mit $|A| = 2^L$ und $p_i = 2^{-L}$ gilt

$$H(X) = -\sum_{i=1}^{2^L} 2^{-L} \log 2^{-L} = L \sum_{i=1}^{2^L} 2^{-L} = L.$$

Unsere Intuition, dass hier eine Codierung mit konstanter Wortlänge L optimal ist (also die kleinstmögliche mittlere Wortlänge erreicht), war also richtig, da nach Satz 5.17 *jeder* Präfixcode $l_E \geq H(X) = L$ erfüllt. Da $-\log p_i = L$ eine ganze Zahl ist, sind für diesen Spezialfall auch Shannon-Codes optimal. ◁

Aufgabe 5.21 Zeigen Sie: Sei E ein Shannon-Code für eine Quelle mit Alphabet A, der für die Verteilung P_X konstruiert wurde. Hat die Quelle in Wirklichkeit die Verteilung P_Y, ändert sich die erwartete Länge im Vergleich zu $H(X)$ um $D_{\mathrm{KL}}(P_Y \parallel P_X)$. ◁

5.1.4. Mehr über Entropie

Die in Satz 5.17 aufgetauchte Entropie einer Quelle (A, X) als untere Schranke der mittleren Codierungslänge wird anschaulich als „mittlere Ungewissheit“ über den Wert von X oder den „mittleren Informationszugewinn“ durch Beobachtung des zufälligen Buchstabens X interpretiert. Laut Satz 5.17 können wir die in X enthaltene Information mit durchschnittlich etwa $H(X)$ bit abspeichern; die Entropie wird deshalb oft in der Einheit Bit gemessen.

Fasst man ein Bit als Antwort einer ja/nein-Frage auf, entspricht $H(X)$ der Anzahl von ja/nein-Fragen, die man (bei optimaler Fragestrategie) stellen muss, um den Wert der Zufallsvariablen X zu erfragen (wir haben hier ignoriert, dass Satz 5.17 nur einen Code garantiert, dessen erwartete Länge bis zu 1 Bit pro Buchstabe über $H(X)$ liegt; wie gesagt, werden wir dieses Extra-Bit später noch loswerden).

Beispiel 5.22 Eine Quelle generiere Zahlen aus $A = \{0, 1, 2, 3\}$ mit unbekannter Wahrscheinlichkeitsverteilung. Um diese Daten binär zu codieren, ist es naheliegend, jedem $a \in A$ seine Binärdarstellung zuzuweisen, so dass jeder Buchstabe zu 2 bit codiert wird (siehe Abbildung 5.5 links, Code E_1); dies entspricht den beiden Fragen „ist $X \geq 2$“ und „ist $X \equiv 1 \pmod 2$“.

Hat man nun Wahrscheinlichkeiten gegeben, etwa $p_0 = p_1 = \frac{1}{8}$, $p_2 = \frac{1}{4}$ und $p_3 = \frac{1}{2}$, ergibt sich $H(X) = 1{,}75 < 2$. Tatsächlich findet man einen Präfixcode, dessen mittlere Wortlänge diesen Wert exakt erreicht (siehe E_2, Abbildung 5.5). Den Code E_2 können wir als Fragestrategie 1. ist $X \in \{0, 1, 2\}$?, 2. wenn ja: $X \in \{0, 1\}$, 3. wenn ja: ist $X = 0$? auffassen.

$a \in A$	$E_1(a)$	$E_2(a)$
0	00	111
1	01	110
2	10	10
3	11	0

Abbildung 5.5.: Zwei mögliche Präfixcodes für die Zahlen $A = \{0, 1, 2, 3\}$ aus Beispiel 5.22. E_1: „naive“ Codierung durch die Binärdarstellung; E_2: optimaler Präfixcode für die im Beispiel gegebene Verteilung mit $l_E = H(X)$.

Die Bedeutung von E_1 erschließt sich mit Bemerkung 5.20: E_1 hätte bei Gleichverteilung von X die kürzeste mittlere Länge. Die Quelle mit den oben gegebenen Wahrscheinlichkeit erzeugt also pro Symbol *weniger* Information als eine gleichverteilte.

Es fällt auf, dass für beide Codes bzw. Fragestrategien E_1 und E_2 die Antworten „ja“ und „nein“ immer gleich wahrscheinlich sind. Am Binärbaum veranschaulicht heißt das, dass für jeden inneren Knoten am rechten und linken Teilbaum jeweils gleich viel „Wahrscheinlichkeitsmasse“ hängt. Tatsächlich ist es eine Eigenschaft optimaler Codes, dass die codierten Bitfolgen möglichst nah an eine Folge unabhängig voneinander gleichverteilter Zufallsbits herankommen. Anschaulich lässt sich das so erklären: wäre die Ausgabe Y *nicht* gleichverteilt, wäre $H(Y) < 1$, so dass wir sie mit einem nachgeschalteten zweiten Code noch sparsamer codieren könnten.◁

Allgemein können wir jedes Alphabet mit $A = \{a_1, \dots, a_s\}$ durch die Binärdarstellung der Zahlen $0, \dots, s-1$ codieren (wobei alle Codewörter gegebenenfalls durch führende Nullen auf die gleiche Länge $l_i = \text{size}(s-1) \approx \log s$ gebracht werden). Dann gilt $l_E = \text{size}(s-1)$ unabhängig von der Verteilung von X. In der Tat ist $\log s$ eine obere Schranke für die Entropie:

Lemma 5.23 *Sei X eine Zufallsvariable mit Alphabet $A = \{a_1, \dots, a_s\}$. Dann gilt*

$$H(X) \leq \log s$$

mit Gleichheit genau dann, wenn X gleichverteilt ist. ◁

Beweis. Wähle in Lemma 5.19 P_Y als Gleichverteilung, also $q_i = \frac{1}{s}$, so ergibt sich nach (5.3)

$$H(X) \leq -\sum_{i=1}^{s} p_i \log \frac{1}{s} = \log s \sum_{i=1}^{s} p_i = \log s$$

mit Gleichheit genau für $p_i = \frac{1}{s}$ für alle $i = 1, \dots, s$. □

Bemerkung 5.24 Die in Definition 5.15 aufgetauchte Funktion $h\colon A \to \mathbb{R}$, $h(a_i) = -\log p_i$ wird auch als *Informationsgehalt* oder *Überraschungswert* des Ereignisses $\{X = a_i\}$ bezeichnet. Seltenere Ereignisse haben demnach einen höheren Überraschungswert, und die Entropie $H(X) = \mathbb{E}[h(X)]$ entspricht dem erwarteten Informationsgehalt. Abbildung 5.6 zeigt links den Überraschungswert in Abhängigkeit von p_i und rechts die Entropie $H_2(p_1) = -p_1 \log p_1 - (1 - p_1) \log(1 - p_1)$ einer binären Zufallsvariable mit Alphabet $A = \{a_1, a_2\}$ in Abhängigkeit von p_1; H_2 wird auch *binäre Entropiefunktion* genannt. ◁

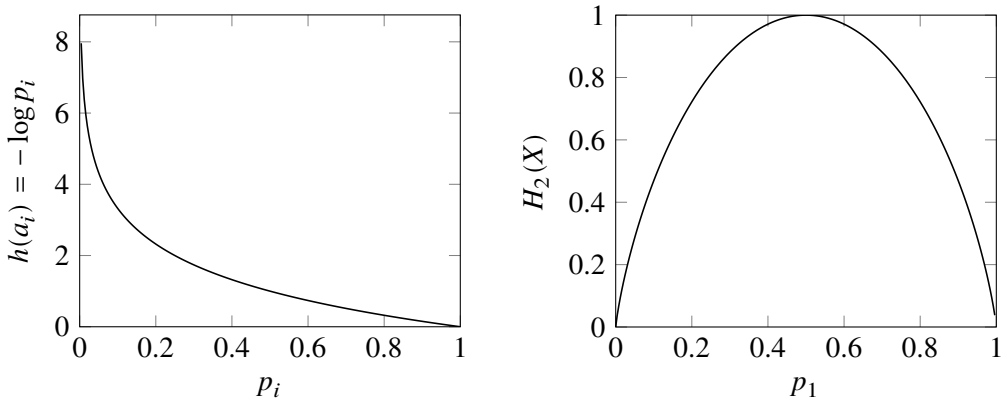


Abbildung 5.6.: Der Überraschungswert $h(a_i)$ (links) ist für unwahrscheinliche Ereignisse größer als für wahrscheinliche. Die binäre Entropiefunktion $H_2(X)$ (rechts) ist symmetrisch mit einem Maximum für $P_X(a_1) = p_1 = 0{,}5$: Bei einer fairen Münze ist der Ausgang eines Münzwurfs am wenigsten vorhersehbar, nach Beobachtung des Ergebnisses weiß man also deutlich mehr als vorher (es wurde „viel Ungewissheit“ beseitigt). In den Extremfäll $p_1 \in \{0, 1\}$ weiß man bereits vorher wie das Experiment ausgeht; durch Beobachtung erfährt man nichts neues, die Entropie ist 0.

Definition 5.25 (Gemeinsame Entropie) Die Entropie einer mehrdimensionalen Zufallsvariable $(X_1, \dots, X_N)$ schreiben wir $H(X_1, \dots, X_N)$ und nennen sie *gemeinsame Entropie* der Zufallsvariablen $X_1, \dots, X_N$. Nach Definition 5.15 ist

$$H(X_1, \dots, X_N) = -\sum_{x_1 \in A_1} \cdots \sum_{x_N \in A_N} P_{X_1 \cdots X_N}(x_1 \cdots x_N) \log P_{X_1 \cdots X_N}(x_1 \cdots x_N),$$

wobei A_i das Alphabet von X_i ist. ◁

Anschaulich gesehen ist $H(X_1, \dots, X_N)$ der Informationszugewinn bei Beobachtung *aller* Zufallsvariablen $X_1, \dots, X_N$. Summieren wir stattdessen die einzelnen Entropien $H(X_1) + \cdots + H(X_N)$, kann das Ergebnis intuitiv nicht kleiner sein als $H(X_1, \dots, X_N)$, da wir ja auch so sämtliche in den Zufallsvariablen enthaltene Information haben. Der zweite Ausdruck kann aber größer sein, nämlich dann, wenn $X_1, \dots, X_N$ nicht unabhängig sind, die Werte von manchen also bereits Information über andere enthalten, was dann beim Summieren mehrfach gezählt wird.

Lemma 5.26 *Für eine N-dimensionale Zufallsvariable $(X_1, \dots, X_N)$ gilt*

$$H(X_1, \dots, X_N) \leq \sum_{i=1}^{N} H(X_i).$$

Gleichheit gilt genau dann, wenn alle X_i paarweise unabhängig sind. ◁

Beweis. Wir zeigen die Aussage für $N = 2$; die Verallgemeinerung folgt dann durch Induktion. Seien also X und Y Zufallsvariablen mit Alphabeten A und B. Nach (1.1) gilt für $a \in A$: $P_X(a) = \sum_{b\in B} P_{XY}(a,b)$. Damit können wir

$$H(X) = -\sum_{a\in A} P_X(a)\log P_X(a) = -\sum_{a\in A}\sum_{b\in B} P_{XY}(a,b)\log P_X(a)$$

schreiben, ersetzen also nur das erste $P_X(a)$ in der Summe. Mit der analogen Umformung von $H(Y)$ gilt

$$\begin{aligned} H(X)+H(Y)-H(X,Y) &= -\sum_{a,b} P_{XY}(a,b)\log P_X(a) \\ &\quad -\sum_{a,b} P_{XY}(a,b)\log P_Y(b) \\ &\quad +\sum_{a,b} P_{XY}(a,b)\log P_{XY}(a,b) \\ &= \sum_{a,b} P_{XY}(a,b)\log\frac{P_{XY}(a,b)}{P_X(a)P_Y(b)} = D_{\mathrm{KL}}(P_{XY}\parallel P_{X'Y'}), \end{aligned}$$

wobei $P_{X'Y'}(a,b) := P_X(a)\cdot P_Y(b)$, d. h. (X',Y') ist die 2-dimensionale Zufallsvariable auf $A\times B$, deren Komponenten X' und Y' die *Marginalverteilungen* P_X und P_Y haben (siehe (1.1)), die aber unabhängig sind, was bei (X,Y) nicht garantiert ist. Nach Lemma 5.19 ist der letzte Ausdruck ≥ 0 und $= 0$ genau dann, wenn $P_{XY} = P_{X'Y'}$ gilt, wenn X und Y also tatsächlich unabhängig sind, womit die Behauptung bewiesen ist. □

Bemerkung 5.27 Nach Bemerkung 5.16 ist die Entropie $H(X) = \mathbb{E}[h(X)]$ der Erwartungswert der von X abgeleiteten Zufallsvariable $h(X)$ mit $h(a) = -\log P_X(a)$. Im Beweis von Lemma 5.26 haben wir gesehen, dass $H(X)$ auch als Erwartungswert einer von (X,Y) abgeleiteten Zufallsvariable aufgefasst werden kann: mit $f : A\times B \to \mathbb{R}, f(a,b) = -\log P_X(a)$ ist $H(X) = \mathbb{E}[f(X,Y)]$. ◁

Mit Lemma 5.26 können wir die in Satz 5.17 verbliebene Lücke von bis zu 1 bit pro Buchstabe schließen. Der Trick besteht darin, nicht mehr einzelne Buchstaben von A, sondern ganze Blöcke von N Buchstaben *gemeinsam* zu codieren. Das „Lücken-Bit“ tritt dann nur noch einmal pro N-Block auf und wird mit wachsendem N vernachlässigbar (Abbildung 5.7).

Satz 5.28 (Quellencodierungssatz für Symbolcodes) *Sei (A,X) eine Quelle. Für jedes $\varepsilon > 0$ ist es möglich, die Daten der Quelle mittels eines Präfixcodes so zu codieren, dass die mittlere Codierungslänge pro Buchstabe von A kleiner als $H(X)+\varepsilon$ ist.* ◁

Beweis. Sei $X^N = X_1X_2\cdots X_N$ ein zufälliges N-Wort der Quelle. Wir definieren nun eine neue Quelle (A^N, X^N), indem wir die N-Wörter von (A,X) blockweise zu Buchstaben des neuen Alphabets A^N zusammenfassen (vgl. auch Seite 26). Da die X_i nach Voraussetzung alle unabhängig sind, folgt nach Lemma 5.26

$$H(X^N) = H(X_1)+\cdots+H(X_N) = N\cdot H(X).$$

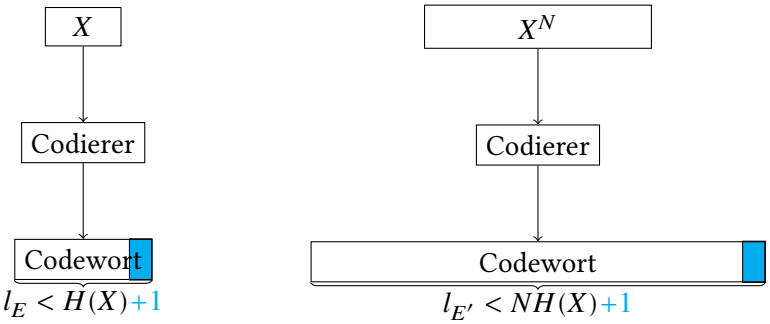


Abbildung 5.7.: Die Idee von Satz 5.28: Durch gemeinsames Codieren von N-Wörtern tritt die Lücke von < 1 Bit nur noch einmal pro N-Block auf, so dass sie umgerechnet pro X weniger als $1/N$ beträgt und damit für $N \to \infty$ gegen 0 geht.

Sei E ein Shannon-Code für die neue Quelle (A^N, X^N), dann gilt nach Satz 5.17

$$N \cdot H(X) \le l_E < N \cdot H(X) + 1 \Leftrightarrow H(X) \le \frac{l_E}{N} < H(X) + \frac{1}{N}.$$

Nun ist l_E die mittlere Codierungslänge eines Buchstabens der geblockten Quelle, also zufälligen N-Worts über A; *pro Buchstabe aus* A erzeugt E im Mittel daher nur l_E/N Bits. Mit $N = 1/\varepsilon$ ist dieser Wert kleiner als $H(X) + \varepsilon$; es folgt die Behauptung. □

Beispiel 5.29 Sei eine Quelle (A, X) mit $A = \{1, 2\}$ und $p_1 = \frac{1}{4}, p_2 = \frac{3}{4}$ gegeben. Ein Shannon-Code E für (A, X) erfüllt $l_1 = \lceil -\log \frac{1}{4} \rceil = 2$ und $l_2 = \lceil -\log \frac{3}{4} \rceil = 1$, und somit $l_E = \frac{1}{4} \cdot 2 + \frac{3}{4} \cdot 1 = \frac{5}{4}$, was deutlich über der Entropie $H(X) \approx 0{,}81$ liegt.

Bemerkung: Dieses Beispiel zeigt, dass Shannon-Codes nicht optimal sein müssen – natürlich wäre es besser, die beiden Symbole mit jeweils Länge 1 zu codieren. Auch damit aber läge die mittlere Wortlänge immer noch deutlich über der Entropie.

Setzen wir $N = 2$, erhalten wir das neue Alphabet $A^2 = \{11, 12, 21, 22\}$ und folgende Wahrscheinlichkeiten und Shannon-Wortlängen für einen Präfixcode E':

$ab \in A^2$	$P_{X_1X_2}(ab)$	$\lceil -\log P_{X_1X_2}(ab) \rceil$
11	$\frac{1}{4} \cdot \frac{1}{4} = \frac{1}{16}$	4
12	$\frac{1}{4} \cdot \frac{3}{4} = \frac{3}{16}$	3
21	$\frac{3}{4} \cdot \frac{1}{4} = \frac{3}{16}$	3
22	$\frac{3}{4} \cdot \frac{3}{4} = \frac{9}{16}$	1

und somit $l_{E'} = \frac{1}{16} \cdot 4 + 2 \cdot \frac{3}{16} \cdot 3 + \frac{9}{16} \cdot 1 = \frac{31}{16}$; die mittlere Länge pro Buchstabe von A sinkt also auf $\frac{31}{32} \approx 0{,}97$ und nähert sich der Entropie $H(X)$ an. ◁

Zur Vorbereitung auf das nächste Kapitel definieren wir noch zwei weitere Entropie-Begriffe und beweisen ein Lemma über diese.

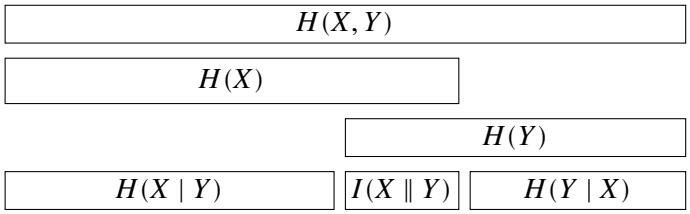


Abbildung 5.8.: Zusammenhang zwischen Einzelentropien, gemeinsamer Entropie, bedingter Entropie und wechselseitiger Information zweier Zufallsvariablen X und Y. Der Überschneidungsbereich $I(X \parallel Y)$ ist genau dann leer, wenn X und Y unabhängig sind.

Definition 5.30 (Bedingte und wechselseitige Entropie) Seien X und Y Zufallsvariablen mit Alphabeten A und B. Für $a \in A$ definieren wir die Entropie von Y, gegeben $\{X = a\}$, als

$$H(Y \mid X = a) := -\sum_{b \in B} P_{Y|X=a}(b) \log P_{Y|X=a}(b)$$

und die *bedingte Entropie* von Y, gegeben X, als

$$H(Y \mid X) := \sum_{a \in A} P_X(a) H(Y \mid X = a) = -\sum_{a \in A} \sum_{b \in B} P_X(a) P_{Y|X=a}(b) \log P_{Y|X=a}(b).$$

Die *wechselseitige Information* von X und Y ist

$$I(X \parallel Y) := H(Y) - H(Y \mid X).$$ ◁

Interpretiert wird $H(Y \mid X)$ als die mittlere Ungewissheit über den Wert von Y, nachdem der Wert von X bekannt ist, und $I(X \parallel Y)$ als Informationsgewinn über Y durch Beobachtung von X („wie viel Ungewissheit wurde von $H(Y)$ durch Beobachtung von X abgezogen"). Damit sind die meisten der folgenden Aussagen intuitiv richtig.

Lemma 5.31 *Sei (X, Y) eine zweidimensionale Zufallsvariable. Dann gelten folgende Aussagen:*

1. $H(X, Y) = H(X) + H(Y \mid X)$,
2. $H(Y \mid X) \leq H(Y)$ *(die Ungewissheit über Y wird nicht größer, wenn wir den Wert von X erfahren),*
3. $I(X \parallel Y) = I(Y \parallel X)$ *(X sagt genauso viel über Y aus wie umgekehrt – diese Beziehung mag überraschen),*
4. $I(X \parallel Y) \geq 0$ *(der Informationsgewinn über eine Zufallsvariable durch Beobachtung einer anderen kann nicht negativ sein).*

Die Beziehung der verschiedenen Entropien sind in Abbildung 5.8 veranschaulicht. ◁

Beweis. Wie üblich seien A und B die Alphabete von X und Y.

1. Es gilt:

$$\begin{aligned} H(Y \mid X) &= -\sum_{a\in A}\sum_{b\in B} \underbrace{P_{Y|X=a}(b)P_X(a)}_{=P_{XY}(a,b)} \log \underbrace{P_{Y|X=a}(b)}_{=P_{XY}(a,b)/P_X(a)} \\ &= -\sum_{a\in A}\sum_{b\in B} P_{XY}(a,b)\log\frac{P_{XY}(a,b)}{P_X(a)} \\ &= \mathbb{E}\left[-\log\frac{P_{XY}(X,Y)}{P_X(X)}\right]. \end{aligned}$$

   Nach Definition sind $H(X,Y) = \mathbb{E}[-\log P_{XY}(X,Y)]$ und $H(X) = \mathbb{E}[-\log P_X(X)]$. Da der Erwartungswert linear ist, also $\mathbb{E}[U+V] = \mathbb{E}[U] + \mathbb{E}[V]$ für Zufallsvariablen U und V gilt, erhalten wir:

$$\begin{aligned} H(X) + H(Y \mid X) &= \mathbb{E}[-\log P_X(X)] + \mathbb{E}\left[-\log\frac{P_{XY}(X,Y)}{P_X(X)}\right] \\ &= \mathbb{E}\left[\log P_X(X) - \log\frac{P_{XY}(X,Y)}{P_X(X)}\right] \\ &= \mathbb{E}[-\log P_{XY}(X,Y)] = H(X,Y). \end{aligned}$$

2. $H(Y \mid X) = H(X,Y) - H(X) \leq H(X) + H(Y) - H(X) = H(Y)$, wobei zuerst der eben gezeigte Punkt 1 und dann Lemma 5.26 benutzt wurden.

3. $I(X \parallel Y) = H(Y) - H(Y \mid X) = H(Y) - H(X,Y) + H(X)$ wieder nach Punkt 1. Da $H(X,Y) = H(Y,X)$ ist dieser Term symmetrisch bezüglich X und Y.

4. Folgt direkt aus der Definition von $I(X \parallel Y)$ und Punkt 2. □

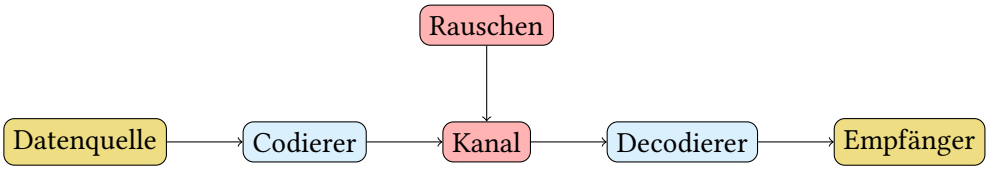
Bemerkung 5.32 Mit

$$\begin{aligned} I(X \parallel Y) &= H(Y \mid X) - H(Y) \\ &= \mathbb{E}\left[-\log\frac{P_{XY}(X,Y)}{P_X(X)} + \log P_Y(Y)\right] \\ &= \mathbb{E}\left[-\log\frac{P_{XY}(X,Y)}{P_X(X)\cdot P_Y(Y)}\right] \end{aligned}$$

haben wir schließlich alle definierten Entropien als Erwartungswerte über (X,Y) dargestellt. Nach (5.2) gilt $I(X \parallel Y) = D_{\mathrm{KL}}(P_X \cdot P_Y \parallel P_{XY})$, der Kullback-Leibler-Abstand zwischen der unabhängigen Verteilung $P_X P_Y$ und der tatsächlichen Verteilung P_{XY}. Anschaulich gesprochen ist die wechselseitige Information also hoch, wenn P_{XY} möglichst „weit“ von einer unabhängigen Verteilung entfernt ist. ◁

5.2. Kommunikation über verrauschte Kanäle / Kanalcodierung

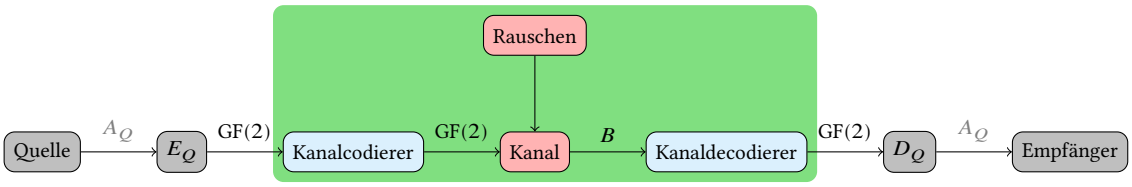
In diesem Kapitel gehen wir nicht mehr von fehlerfreier Kommunikation aus, sondern berücksichtigen *Rauschen*, welches die Daten während der Übertragung über den Kanal beeinflusst, haben als wieder unser ursprüngliches Setup:



Eine beliebige Quelle (A_Q, X_Q) können wir nach Abschnitt 5.1 mit einem Symbolcode E_Q in eine Folge von Bits (Alphabet GF(2)) codieren. Wir nehmen deshalb in diesem Kapitel an, dass eine solche Quellencodierung bereits stattgefunden hat (und auf Empfängerseite ein entsprechender Decodierer D_Q existiert), und teilen das Codieren / Decodieren in zwei Bereiche auf:

- Die *Quellencodierung* sorgt dafür, dass die Buchstaben der Quelle möglichst sparsam in eine Folge von Bits übersetzt werden. Dieses Problem haben wir in Abschnitt 5.1 behandelt und gelöst.
- Die *Kanalcodierung* sorgt dafür, dass der Empfänger diese Bitfolge (mit hoher Wahrscheinlichkeit) rekonstruieren kann, auch wenn beim Verschicken Übertragungsfehler auftreten.

Bei der Kanalcodierung dürfen wir also oBdA von einer binären Quelle ausgehen und uns auf den in folgender Grafik grün umrahmten Bereich konzentrieren:



Wir spezifizieren zunächst unser Kanalmodell, um dann zu sehen, wie wir durch Codierung trotz Rauschen eine beliebig kleine Fehlerwahrscheinlichkeit erreichen können. Der Einfachheit halber beschränken wir uns auf Kanäle, deren Eingabe ebenfalls aus Bits besteht.

5.2.1. Kanalmodell und Kapazität

In Beispiel 4.33 haben wir bereits einen Kanal verwendet, ohne genau zu definieren was das überhaupt ist. Dieser hat ein gesendetes Bit mit Wahrscheinlichkeit $\varepsilon = \frac{1}{4}$ falsch übertragen. Allgemein wollen wir einen Kanal dadurch modellieren, dass man ihn mit einer 0 oder 1 füttern kann, und abhängig von dieser Eingabe eine Wahrscheinlichkeitsverteilung der möglichen Ausgaben erhält.

Definition 5.33 (Kanal) Ein *Kanal* (mit binärer Eingabe) ist definiert durch das Tripel

$$\mathcal{K} = (B, P_{\mathcal{K},0}, P_{\mathcal{K},1})$$

mit dem *Ausgabealphabet* B sowie zwei Wahrscheinlichkeitsverteilungen $P_{\mathcal{K},0}$ und $P_{\mathcal{K},1}$, deren Alphabet jeweils B ist. Für $x \in \mathrm{GF}(2)$ und $y \in B$ gibt die *Übertragungswahrscheinlichkeit*

$P_{\mathcal{K},x}(y)$ an, mit welcher Wahrscheinlichkeit der Kanal das Symbol y ausgibt, wenn x gesendet wurde. ◁

Bemerkung 5.34 Analog zur Quelle gehen wir davon aus, dass der Kanal beliebig oft hintereinander genutzt werden kann und die gesendeten Bits jeweils unabhängig voneinander und immer mit den gleichen Übertragungswahrscheinlichkeiten beeinflusst (man nennt solche Kanäle *gedächtnislos*). Dies ist natürlich eine Vereinfachung: Der Handyempfang etwa kann je nach Wetterlage oder Gebäuden zwischen Gerät und Sendemast, generell mit Zeit und Ort variieren. ◁

Beispiel 5.35 Der im Eingangsbeispiel (Beispiel 4.33) verwendete Kanal wird *binärer symmetrischer Kanal mit Fehlerwahrscheinlichkeit* ε oder kurz $\mathrm{BSC}(\varepsilon)$ (für *binary symmetric channel*) genannt, wobei $\varepsilon \in \left(0, \frac{1}{2}\right)$ vorausgesetzt ist. Er ist durch das Tripel

$$\mathrm{BSC}(\varepsilon) = (\mathrm{GF}(2), P_{\mathrm{BSC}(\varepsilon),0}, P_{\mathrm{BSC}(\varepsilon),1})$$

definiert, hat also Ausgabealphabet $B = \mathrm{GF}(2)$, und für die Eingabe $x \in \{0,1\}$ ist die Verteilung der Ausgabe $y \in \{0,1\}$ gegeben als

$$P_{\mathrm{BSC}(\varepsilon),x}(y) = \begin{cases} 1-\varepsilon & \text{wenn } x = y \\ \varepsilon & \text{wenn } x \neq y. \end{cases}$$

$\mathrm{BSC}(\varepsilon)$ überträgt ein Bit mit Wahrscheinlichkeit $1-\varepsilon$ korrekt und mit Wahrscheinlichkeit ε falsch. Für $\varepsilon = \frac{1}{2}$ wäre die Ausgabe unabhängig von der Eingabe und der Kanal damit nutzlos, für $\varepsilon > \frac{1}{2}$ ist es sinnvoller, die Ausgabe umzudefinieren ($0 \to 1$ und $1 \to 0$), um $\varepsilon < \frac{1}{2}$ zu erhalten.

Der *binäre Auslöschungskanal* oder kurz $\mathrm{BEC}(\varepsilon)$ (*binary erasure channel*) mit $\varepsilon \in (0,1)$ hat das Ausgabealphabet $B = \{0, 1, ?\}$ und Übertragungswahrscheinlichkeiten

$$P_{\mathrm{BEC}(\varepsilon),x}(y) = \begin{cases} 1-\varepsilon & \text{für } x = y \\ \varepsilon & \text{für } y = ? \\ 0 & \text{sonst} \end{cases}$$

für alle $x \in \{0,1\}$ $y \in \{0,1,?\}$. Er macht zwar nie aus einer 0 eine 1 oder umgekehrt, dafür geht mit Wahrscheinlichkeit ε sämtliche Information über ein gesendetes Bit verloren, was mit der Ausgabe „?“ markiert wird. Beide Kanäle sind in Abbildung 5.9 dargestellt. ◁

Bemerkung 5.36 Die Definition eines Kanals $\mathcal{K}$ umfasst nur die Verteilung der Ausgabe bei *gegebener* Eingabe $x \in \mathrm{GF}(2)$. Wählen wir nun auch den Input zufällig gemäß einer Verteilung P_X mit Alphabet $\mathrm{GF}(2)$, erhalten wir eine zweidimensionale Zufallsvariable (X, Y), deren X-Komponente den (binären) Kanalinput und deren Y-Komponente den Kanaloutput (Alphabet B) angibt. Mit $P_{Y|X=x}(y) = P_{\mathcal{K},x}(y)$ ist die gemeinsame Verteilung

$$P_{XY}(x,y) = P_X(x) P_{Y|X=x}(y) = P_X(x) P_{\mathcal{K},x}(y),$$

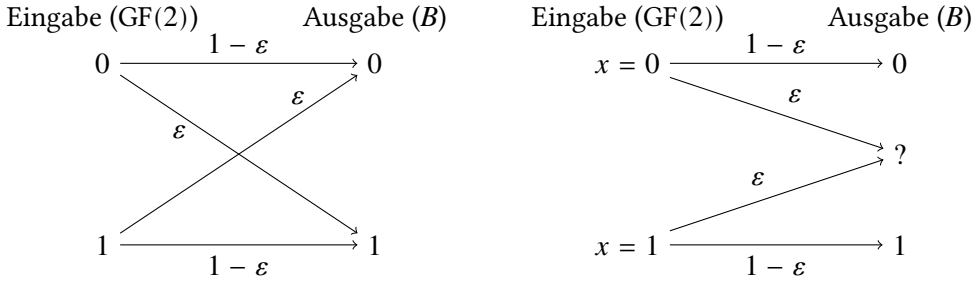


Abbildung 5.9.: Binärer symmetrischer Kanal (links) und binärer Auslöschungskanal (rechts). Die Werte an den Pfeilen geben die jeweiligen Übertragungswahrscheinlichkeiten an.

$P_{XY}(x,y)$ gibt die Wahrscheinlichkeit an, dass x gesendet *und* y empfangen wurde. Die Marginalverteilung von Y ist nach (1.1)

$$P_Y(y) = \sum_{x \in \mathrm{GF}(2)} P_X(x) P_{Y|X=x}(y).$$

Nach dem Satz von Bayes (Aufgabe 1.27) kann der Empfänger nach Beobachtung der Kanalausgabe $\{Y = y\}$ die (durch y bedingte) Verteilung der Eingabe X berechnen:

$$P_{X|Y=y}(x) = \frac{P_{Y|X=x}(y) P_X(x)}{P_Y(y)} = \frac{P_{Y|X=x}(y) P_X(x)}{\sum_{x' \in \mathrm{GF}(2)} P_{Y|X=x'}(y) P_X(x')}$$

Gehen wir davon aus, dass dem Empfänger die Eingangsverteilung P_X bekannt ist, kann er wegen $P_{Y|X=x}(y) = P_{\mathscr{K},x}(y)$ den Ausdruck auf der rechten Seite berechnen und somit $P_{X|Y=y}(x)$ bestimmen. ◁

Beispiel 5.37 Beim BSC(0,15) sei die Eingangsverteilung $P_X(0) = 0{,}9$, $P_X(1) = 0{,}1$ gegeben. Der Empfänger beobachte $\{Y = 1\}$. Dann gilt

$$P_{X|Y=1}(1) = \frac{P_{Y|X=1}(1) P_X(1)}{\sum_{x' \in \mathrm{GF}(2)} P_{Y|X=x'}(1) P_X(x')} = \frac{0{,}85 \cdot 0{,}1}{0{,}85 \cdot 0{,}1 + 0{,}15 \cdot 0{,}9} = \frac{0{,}085}{0{,}22} \approx 0{,}39.$$

Selbst nach dem Empfang einer 1 ist die Wahrscheinlichkeit, dass eine 0 gesendet wurde, mit $P_{X|Y=1}(0) \approx 1 - 0{,}39 = 0{,}61$ immer noch größer als die der 1 – wenn auch nicht mehr so groß wie vor der Beobachtung ($P_X(0) = 0{,}9 > P_{X|Y=1}(0) \approx 0{,}61$). ◁

Durch Beobachtung des Outputs gewinnen wir also Information über den gesendeten Buchstaben. Nach den Überlegungen des letzten Kapitels können wir sogar quantisieren, wie viel Information uns die Beobachtung von Y über X gibt, nämlich gerade $I(X \parallel Y)$. Der Wert $I(X \parallel Y)$ hängt von der Verteilung P_X ab; durch Veränderung dieser können wir die vom Kanal übertragene Information variieren. Dies führt zu folgender Definition.

Definition 5.38 (Kanalkapazität) Sei ein Kanal $\mathcal{K}$ gegeben. Für $p_0 \in [0,1]$ sei $X^{(p_0)}$ eine Zufallsvariable mit Alphabet GF(2) und $P_{X^{(p_0)}}(0) = p_0$, $P_{X^{(p_0)}}(1) = 1 - p_0$. Weiterhin sei $\left(X^{(p_0)}, Y^{(p_0)}\right)$ das nach Bemerkung 5.36 durch $X^{(p_0)}$ und $\mathcal{K}$ bestimmte Paar aus Kanalinput und -Output. Die *Kapazität* von $\mathcal{K}$ ist definiert als

$$C \coloneqq \max_{p_0 \in [0,1]} I\left(X^{(p_0)} \parallel Y^{(p_0)}\right), \tag{5.4}$$

also die über alle Eingangsverteilungen maximierte wechselseitige Information zwischen In- und Output des Kanals. ◁

Beispiel 5.39 (Kapazität des BSC) Wir wollen die Kapazität des binären symmetrischen Kanals BSC(ε) berechnen. Dazu schreiben wir $I(X \parallel Y) = H(Y) - H(Y \mid X)$ und berechnen zunächst $H(Y \mid X)$: Nach Definition ist

$$\begin{aligned} H(Y \mid X) &= \sum_{a \in \{0,1\}} P_X(a) H(Y \mid X = a) \\ &= -\sum_{a \in \{0,1\}} P_X(a) \sum_{b \in \{0,1\}} P_{Y|X=a}(b) \log P_{Y|X=a}(b) \end{aligned}$$

Da beim BSC der Ausdruck $P_{Y|X=a}(b)$ entweder ε (für $a \neq b$) oder $1 - \varepsilon$ (für $a = b$) ist und die Summe über beide Fälle läuft, gilt

$$\begin{aligned} &= -\sum_{a \in \{0,1\}} P_X(a) \left((1-\varepsilon)\log(1-\varepsilon) + \varepsilon \log \varepsilon\right) \\ &= -(1-\varepsilon)\log(1-\varepsilon) - \varepsilon \log \varepsilon = H_2(\varepsilon) \end{aligned}$$

mit der binären Entropiefunktion H_2 aus Bemerkung 5.24; insbesondere ist $H(Y \mid X)$ also *unabhängig* von der Eingabeverteilung P_X und spielt deshalb für die Maximierung in (5.4) keine Rolle, es gilt nur $H(Y)$ zu maximieren. Nach Lemma 5.26 ist $H(Y) \leq \log |B| = \log 2 = 1$ mit Gleichheit genau dann, wenn Y gleichverteilt ist. Wir zeigen, dass bei gleichverteiltem X auch Y gleichverteilt ist, also $H(Y)$ und damit auch $I(X \parallel Y)$ maximal werden.

Sei also $P_X(0) = P_X(1) = \frac{1}{2}$. Dann gilt

$$\begin{aligned} P_Y(0) &= \sum_{a \in \{0,1\}} P_{XY}(a, 0) \\ &= \sum_{a \in \{0,1\}} P_X(a) P_{Y|X=a}(0) \\ &= \frac{1}{2}\left(P_{Y|X=0}(0) + P_{Y|X=1}(0)\right) \\ &= \frac{1}{2}(1 - \varepsilon + \varepsilon) = \frac{1}{2} \end{aligned}$$

und damit auch $P_Y(1) = 1 - P_Y(0) = \frac{1}{2}$. Gleichverteiltes X maximiert also tatsächlich $H(Y)$, und die Kapazität des BSC berechnet sich zu

$$C = H(Y) - H(Y \mid X) = 1 - H_2(\varepsilon).$$

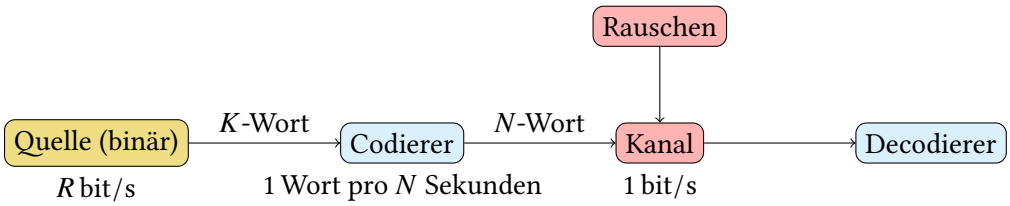
Bei Betrachtung der Extremfälle entspricht dies der Erwartung: ist $\varepsilon = 0$, passieren keine Fehler und die übertragene Informationsmenge entspricht einem Bit, also genau der Eingabe. Für $\varepsilon = \frac{1}{2}$ ist die Ausgabe von der Eingabe komplett unabhängig; es wird keinerlei Information übertragen.

In Beispiel 5.37 mit $\varepsilon = 0{,}15$ erhalten wir $H(Y \mid X) \approx 0{,}61$. Außerdem haben wir dort für $P_X(0) = 0{,}1$ bereits $P_Y(1) \approx 0{,}22$ berechnet, so dass $H(Y) = H_2(0{,}22) \approx 0{,}76$ folgt und damit $I(X \parallel Y) = H(Y) - H(Y \mid X) \approx 0{,}15$ ist. Die bei gleichverteiltem X erreichte Kapazität des BSC(0,15) hingegen haben wir eben zu $1 - H_2(0{,}15) \approx 0{,}39$ berechnet. Durch die ungünstige Eingangsverteilung in Beispiel 5.37 wird also sehr viel weniger Information übertragen als eigentlich möglich wäre. ◁

5.2.2. Blockcodes

Die bisher betrachteten Symbolcodes waren darauf optimiert, die mittlere Codierungslänge bei fehlerfreier Übertragung zu minimieren. Zur Absicherung der Übertragung auf einem fehlerhaften Kanal verwenden wir stattdessen *Blockcodes* mit konstanter Codierungslänge, die im Folgenden hergeleitet werden.

Unsere binäre Quelle (vgl. 99) erzeuge $R \leq 1$ Bits pro Sekunde (durch Skalierung der Zeiteinheit können wir hier natürlich jede beliebige Rate annehmen; ersetzen Sie einfach „Sekunde“ durch „Zeiteinheit“). Um diese über einen binären Kanal zu verschicken, der maximal ein Bit pro Sekunde annimmt, warten wir zunächst N Sekunden, in denen die Quelle $K := RN$ Buchstaben, also ein binäres K-Wort erzeugt, von denen es insgesamt 2^K gibt (wir gehen in dieser Betrachtung davon aus, dass $K \in \mathbb{N}$ ist). Dieses K-Wort codieren wir zu einem binären N-Wort, haben es also mit *konstanten* Codewortlängen zu tun.



Da unser Kanal nach Annahme 1 bit/s überträgt, kann ein codiertes N-Wort innerhalb von N Sekunden verschickt werden. Innerhalb dieser N Sekunden hat die Quelle ein neues K-Wort erzeugt, welches dann sofort codiert und in den nächsten N Sekunden übertragen wird. Der Codierer muss also K Bits zwischenspeichern können; darüber hinaus gibt es aber keinen „Rückstau“. Abgesehen von der codierungs-basierten Verzögerung von N Sekunden können alle Daten übertragen werden. Dem Verhältnis K/N kommt dabei die Rolle der schon in Beispiel 4.33 eingeführten *Codierungsrate* zu, die wir ab jetzt nur noch *Rate* nennen werden.

Definition 5.40 (Blockcode) Seien $N, K \in \mathbb{N}$. Ein (N,K)-*Blockcode* ist ein Tupel $(E_{\mathscr{C}}, \mathscr{C})$, bestehend aus einer Abbildung

$$E_{\mathscr{C}} \colon \mathrm{GF}(2)^K \to \mathrm{GF}(2)^N,$$

also einer Vorschrift die jedem binären K-Wort ein binäres N-Wort zuweist, sowie der als Bild von $E_{\mathscr{C}}$ definierten Familie von 2^K Codewörtern

$$\mathscr{C} = \{E_{\mathscr{C}}(w)\}_{w \in \mathrm{GF}(2)^K}.$$

N wird *Blocklänge* von $\mathscr{C}$ genannt; die *Rate* des Codes ist definiert als $R = K/N = \frac{\log|\mathscr{C}|}{N}$. ◁

Bemerkung 5.41 Die durch $E_{\mathscr{C}}$ gegebene Zuordnung der von der Quelle erzeugten K-Wörter zu Codewörtern ist für die fehlerkorrigierenden Eigenschaften eines Codes nicht relevant; es kommt nur auf die Struktur der Codewörter $\mathscr{C}$ an. Wir werden deshalb in Zukunft das $E_{\mathscr{C}}$ gar nicht mehr erwähnen und einfach von $\mathscr{C}$ als „dem Code“ reden. Dass wir diesen als *Familie* und nicht als Menge definieren, liegt daran, dass wir (aus technischen Gründen, die später im Beweis klar werden) keine Injektivität von $E_{\mathscr{C}}$ fordern, weshalb $\mathscr{C}$ auch mehrfache Einträge enthalten kann. ◁

Beispiel 5.42 In Beispiel 4.33 haben wir den 3-Wiederholungscode $E \colon \mathrm{GF}(2) \to \mathrm{GF}(2)^3$ definiert und damit ohne es zu wissen bereits einen Blockcode genutzt: dabei war $E_{\mathscr{C}} = E$ gegeben durch $E_{\mathscr{C}}(a) = aaa$ für $a \in \mathrm{GF}(2)$. Damit ergibt sich $K = 1$ und die Blocklänge $N = 3$. Die Rate des Codes ist $K/N = 1/3$, und $\mathscr{C} = \{E(0), E(1)\} = \{000, 111\}$. ◁

Bemerkung 5.43 Schicken wir ein Codewort $x = x_1 \cdots x_N \in \mathscr{C}$ aus einem Blockcode $\mathscr{C}$ Bit für Bit über einen Kanal $\mathscr{K} = (B, P_{\mathscr{K},0}, P_{\mathscr{K},1})$, erhalten wir auf Empfängerseite die Ausgabe $y = y_1 \cdots y_N \in B^N$ mit Wahrscheinlichkeit

$$\prod_{i=1}^{N} P_{\mathscr{K},x_i}(y_i),$$

da wir ja (Bemerkung 5.34) davon ausgehen, dass der Kanal aufeinanderfolgende Bits unabhängig und gleich behandelt. ◁

Erhält der Empfänger ein N-Wort $y \in B^N$, muss er sich für ein Codewort $x \in \mathscr{C}$ entscheiden, von dem er „glaubt“, dass es gesendet wurde. Dieser Entscheider wird *Decodierer* genannt.

Definition 5.44 Ein *Decodierer* für einen (N,K)-Blockcode $\mathscr{C}$ und einen Kanal $\mathscr{K}$ mit Ausgabealphabet B ist eine Abbildung

$$D_{\mathscr{C}} \coloneqq B^N \to \mathscr{C} \cup \{\mathsf{e}\}.$$

Der Decodierer entscheidet sich also bei gegebenem Kanaloutput $y \in B^N$ entweder für ein Codewort aus $\mathscr{C}$ oder nutzt das spezielle Symbol e um einen Fehlschlag anzuzeigen.

Wird $x \in \mathscr{C}$ gesendet, $y \in B^N$ empfangen und ist $D_{\mathscr{C}}(y) \neq x$, spricht man von einem *Decodierfehler*. Bei $D_{\mathscr{C}}(y) = \mathsf{e}$ tritt auf jeden Fall ein Decodierfehler auf. Aber auch bei $D_{\mathscr{C}}(y) \in \mathscr{C}$ kann ein Decodierfehler vorliegen, nämlich dann wenn $D_{\mathscr{C}}$ ein *anderes* als das gesendete Codewort ausgibt. ◁

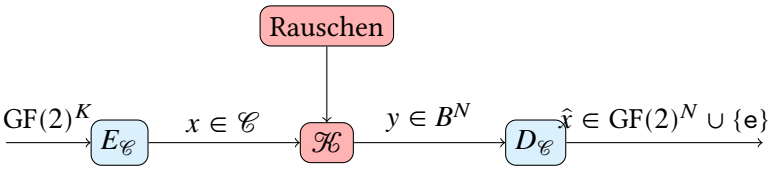
Beispiel 5.45 Der auf Mehrheitsentscheidung basierende Decodierer im Eingangsbeispiel (Beispiel 4.33) für den 3-Wiederholungscode ist durch

$$D_{\mathscr{C}}(y_1y_2y_3) = \begin{cases} 000 & \text{falls } |\{i \in \{1,2,3\} : y_i = 0\}| \geq 2 \\ 111 & \text{sonst} \end{cases}$$

für $y \in B^N = \mathrm{GF}(2)^3$ definiert; er entscheidet sich für das Codewort, das der Ausgabe y „am nächsten“ ist, und liegt damit bei höchstens einem Übertragungsfehler pro Codewort richtig. Er gibt immer ein Codewort aus, nutzt also nicht das Fehlschlags-Symbol e. Dieses könnte beispielsweise beim 4-Wiederholungscode sinnvoll sein: passieren bei einem 4-Wort *aaaa* genau 2 Übertragungsfehler, herrscht Gleichstand zwischen 0en und 1en, so dass sich der Decodierer für keine Mehrheit entscheiden kann.

Was ein Empfänger beim Empfang von e macht, hängt von den Anforderungen der jeweiligen Anwendung ab und ist nicht Teil unserer Betrachtungen. Er könnte zum Beispiel eine Wiederholung der Übertragung anfragen (etwa beim Download eines Programms) oder aber sich damit zufrieden geben, dass ein paar Daten fehlen (Knackser beim Telefonieren). ◁

Mit Blockcode, Kanal und Decodierer erhalten wir nun folgendes Nachrichtenübertragungssystem:



Dabei fehlt eigentlich rechts noch eine Komponente, die das von $D_{\mathscr{C}}$ ausgegebene Codewort $\hat{x} \in \mathscr{C}$ wieder in das ursprüngliche Wort aus $\mathrm{GF}(2)^K$ (also die Eingabe von $E_{\mathscr{C}}$) übersetzt, die wir aber weglassen, da uns wie gesagt die genaue Form von $E_{\mathscr{C}}$ nicht interessiert.

Für ein solches System können wir nun die Wahrscheinlichkeit eines Decodierfehlers bestimmen:

Definition 5.46 Seien ein (N,K)-Blockcode $\mathscr{C}$, ein Kanal $\mathscr{K}$ und ein passender Decodierer $D_{\mathscr{C}}$ gegeben. Für gegebenes $x \in \mathscr{C}$ ist

$$p_{\mathrm{Err}}(x) := \sum_{\substack{y \in B^N : \\ D_{\mathscr{C}}(y) \neq x}} \prod_{i=1}^{N} P_{\mathscr{K},x_i}(y_i)$$

die Wahrscheinlichkeit eines Decodierfehlers, wenn x gesendet wurde. Die Summe läuft dabei über alle $y \in B^N$, für die der Decodierer ein falsches Ergebnis liefert; der Ausdruck in der Summe ist nach Bemerkung 5.43 die Wahrscheinlichkeit, dass dieses y bei gesendetem x auftritt.

Die *mittlere Blockfehlerrate* ist definiert als

$$p_{\text{Err}}(\mathscr{C}) = \frac{1}{|\mathscr{C}|} \sum_{x \in \mathscr{C}} p_{\text{Err}}(x) = \mathbb{E}[p_{\text{Err}}(X)],$$

wenn X eine gleichverteilte Zufallsvariable mit Alphabet $\mathscr{C}$ ist. Schließlich ist der *maximale Blockfehler*

$$p^*_{\text{Err}}(\mathscr{C}) = \max_{x \in \mathscr{C}} p_{\text{Err}}(x). \qquad \triangleleft$$

Beispiel 5.47 In Beispiel 4.33 haben wir $p_{\text{Err}}(x) \approx 0{,}16$ für $x \in \{000, 111\}$ berechnet. Da $p_{\text{Err}}(x)$ somit für alle $x \in \mathscr{C}$ gleich ist, gilt in diesem Fall $p_{\text{Err}}(\mathscr{C}) = p^*_{\text{Err}}(\mathscr{C}) \approx 0{,}16$. $\triangleleft$

Wir haben nun alle Definitionen beisammen, um das wichtigste Resultat der Informationstheorie zu formulieren.

Satz 5.48 (Kanalcodierungssatz) *Sei ein Kanal $\mathscr{K}$ mit Kapazität $C > 0$ gegeben. Für jedes $\varepsilon > 0$ und $0 < R < C$ existiert ein Blockcode $\mathscr{C}$ mit Rate $R' \geq R$ und ein Decodierer $D_{\mathscr{C}}$, so dass für den maximalen Blockfehler dieses Systems $p^*_{\text{Err}}(\mathscr{C}) < \varepsilon$ gilt.* $\triangleleft$

Beispiel 5.49 Wir schließen nun das Eingangsbeispiel (Beispiel 4.33) ab. Der dabei verwendete BSC($\frac{1}{4}$)-Kanal hat nach Beispiel 5.39 die Kapazität $C = 1 - H_2(\varepsilon) \approx 0{,}189 > \frac{1}{6}$. Bereits bei Codierungsrate $R = \frac{1}{6}$, d. h. 6-mal so vielen Kanal- wie Datenbits, könnten wir nach Satz 5.48 eine beliebig kleine Fehlerrate erreichen. Zum Vergleich: der 7-Wiederholungscode R_7 hat bei deutlich niedrigerer Rate $R' = \frac{1}{7}$ noch eine Fehlerwahrscheinlichkeit von $p_{\text{Err}}(R_7) \approx 0{,}07$. $\triangleleft$

Zum Beweis benötigen wir noch den Begriff der *Typizität*, der im folgenden Abschnitt behandelt wird.

5.2.3. Typizität

Schon beim Beweis von Satz 5.28 haben wir gesehen, dass es hilft, möglichst lange Wörter gemeinsam zu codieren. Demselben Prinzip werden wir hier wieder begegnen. Ein Phänomen namens *Typizität* liefert uns eine hübsche theoretische Begründung dafür. Anschaulich besagt es Folgendes: Ist X eine Zufallsvariable mit Alphabet A, gilt $H(X) < \log|A|$ und ist $N \in \mathbb{N}$ „ausreichend groß“, dann gibt es eine Teilmenge $T \subseteq A^N$, so dass

- ein zufälliges N-Wort mit sehr großer Wahrscheinlichkeit in T liegt, aber andererseits
- nur ein *extrem kleiner* Bruchteil aller möglichen N-Wörter in T liegt.

Der Beweis von Satz 5.48 wird im Wesentlichen darauf basieren, sich nur um die korrekte Übertragung solcher „typischer“ N-Wörter zu kümmern. Die erste Eigenschaft von T sorgt dann dafür, dass man die Fehlerrate klein hält, während die zweite für die Behauptung hinsichtlich der Codierungsrate genutzt wird.

Wie definieren wir nun diese Menge T? Sei X eine Zufallsvariable mit Alphabet $\mathrm{GF}(2)$, $P_X(0) = p_0$, $P_X(1) = p_1 = 1 - p_0$. Durch N-fache Wiederholung des Experiments erhalten wir zufällige binäre N-Wörter X^N aus $\mathrm{GF}(2)^N$. Wir sind nun daran interessiert, wie viele 1er in einem solchen Wort $x \in \mathrm{GF}(2)^N$ zu erwarten sind. Sei $r(x) \coloneqq \{i \colon x_i = 1\}$. Für ein bestimmtes x mit $r(x) = \rho$ gilt $P_{X^N}(x) = p_1^\rho p_0^{N-\rho}$ und es gibt insgesamt $\binom{N}{\rho}$ solche $x \in \mathrm{GF}(2)^N$ mit $r(x) = \rho$. Deshalb ist $r(X^N)$ binomialverteilt:

$$P\left(r(X^N) = \rho\right) = \binom{N}{\rho} p_1^\rho p_0^{N-\rho},$$

Aus der Statistik-Vorlesung ist bekannt, dass der Erwartungswert der Binomialverteilung $\mu_{r(X^N)} = Np_1$ und die Standardabweichung $\sigma_{r(X^N)} = \sqrt{Np_0p_1}$ beträgt. Wir erwarten also ungefähr Np_1 Einser in einem N-Wort; darüber hinaus sorgt die Wurzel in $\sigma_{r(X^N)}$ dafür, dass r im Vergleich zu N mit wachsendem N immer weniger streut – anders gesagt, „die meisten“ Wörter $x \in \mathrm{GF}(2)^N$ haben „ungefähr“ Np_1 Einser.

Ist allgemein X eine Zufallsvariable mit Alphabet A, $|A| = s$, enthält X^N typischerweise ungefähr p_iN-mal das Symbol $a_i \in A$ (für diese intuitive Herleitung ignorieren wir die Tatsache, dass p_iN nicht ganzzahlig sein muss). Für ein solches x gilt dann

$$\begin{aligned}
P_{X^N}(x) &= P_X(x_1)P_X(x_2)\cdots P_X(x_N) \\
&\approx \prod_{i=1}^{s} p_i^{p_iN} \\
&= 2^{\log \prod_{i=1}^{s} p_i^{p_iN}} \\
&= 2^{\sum_{i=1}^{s} \log p_i^{p_iN}} \\
&= 2^{N \sum_{i=1}^{s} p_i \log p_i} \\
&= 2^{-NH(X)},
\end{aligned}$$

weshalb wir T als die Menge der Wörter x definieren, für die $P_{X^N}(x)$ nicht zu weit von $2^{-NH(X)}$ entfernt liegt.

Definition 5.50 (Typizität) Sei X eine Zufallsvariable mit Alphabet A, $N \in \mathbb{N}$ und $\beta > 0$. Ein N-Wort $x \in A^N$ heißt *β-typisch für X*, wenn

$$2^{-N(H(X)+\beta)} \leq P_{X^N}(x) \leq 2^{-N(H(X)-\beta)}. \tag{5.5}$$

oder äquivalent

$$\left| -\frac{1}{N} \log P_{X^N}(x) - H(X) \right| \leq \beta$$

gilt. Die Menge aller β-typischen N-Wörter

$$T_{N,\beta} = T_{N,\beta}(X) \coloneqq \left\{ x \in A^N \colon \left| -\frac{1}{N} \log P(x) - H(X) \right| \leq \beta \right\}$$

heißt *(β-)typische Menge von X*. ◁

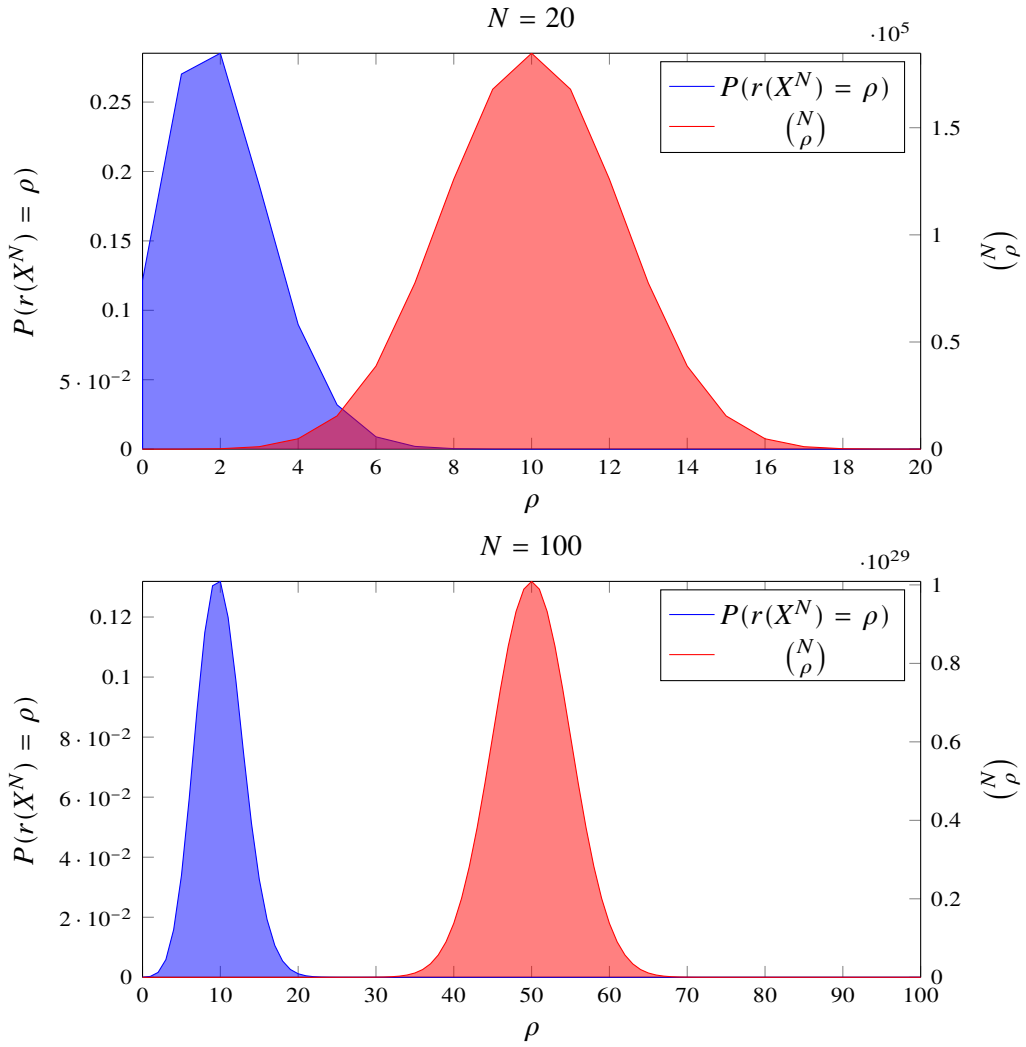


Abbildung 5.10.: Verteilung der Anzahl Einser in einem N-Wort, $r(X^N)$ (blau), sowie Anzahl der N-Wörter $\binom{N}{\rho}$ (rot) für $N = 20$ bzw. $N = 100$ und jeweils $p_1 = 0{,}1$.
Die blaue Fläche repräsentiert die gesamte Wahrscheinlichkeit, die sich mit wachsendem N immer enger um den Erwartungswert von $\mathbb{E}[r(X^N)] = Np_1$ Einsern pro N-Wort konzentriert; die typische Menge enthält gerade die N-Wörter aus diesem Bereich. Die rote Fläche hingegen entspricht der Anzahl der Elemente von A^N, von denen mit wachsendem N ein immer größerer Anteil ungefähr $\rho = \binom{N}{N/2}$ Einser enthält. Wie die Grafik zeigt, wachsen die beiden Spitzen für größere N auseinander: eine verschwindend kleine Menge von N-Wörtern versammelt praktisch die gesamte Wahrscheinlichkeit auf sich.
Die Grafik zeigt auch, was im Fall $H(X) = \log|A|$ schiefgeht: dann liegen die beiden Peaks für jedes N genau aufeinander, der „Auseinanderwachs-Effekt“ geht verloren.

Beispiel 5.51 Sei $A = \{0,1\}$ und $p_0 = \frac{1}{3}$, $p_1 = \frac{2}{3}$. Dann sind 111111000, 000111111 und 110111100 Beispiele für typische 9-Wörter (in diesem Fall sogar für beliebiges β). Das 9-Wort 111111111 ist für hinreichend kleines β nicht typisch, obwohl es unter allen das mit der größten Wahrscheinlichkeit ist.

$T_{N,\beta}$ enthält also im Allgemeinen die wahrscheinlichsten Elemente von A^N gar nicht. Dies fällt nicht ins Gewicht, da es von diesen vergleichsweise wenige gibt: im Beispiel etwa gibt es nur *ein* Wort mit 9 Einsern, aber $\binom{9}{6} = 84$ Wörter mit 6 Einsern.

In Abbildung 5.10 sind für $p_1 = 0{,}1$ die Anzahl der N-Wörter mit ρ Einsen sowie die Wahrscheinlichkeit $P(r(X^N) = \rho)$ für $N = 20$ und $N = 100$ dargestellt, wodurch der in Lemma 5.52 formalisierte „Typizitäts-Effekt“ gut sichtbar wird. ◁

Wir beweisen nun, dass $T_{N,\beta}$ die Bedingungen an die Anfang des Abschnitts beschriebene Menge T erfüllt.

Lemma 5.52 *Sei $\beta > 0$ und X eine Zufallsvariable.*

1. *Zu jedem $\varepsilon > 0$ existiert ein $N_0 \in \mathbb{N}$, so dass für alle $N > N_0$ gilt:*
$$P(X^N \in T_{N,\beta}) > 1 - \varepsilon.$$
2. $\left|T_{N,\beta}\right| \leq 2^{N(H(X)+\beta)}$. ◁

Beweis. 1. Sei A das Alphabet von X, $X^N = X_1 \cdots X_N$ ein zufälliges N-Wort und $h\colon A \to \mathbb{R}$, $h(a_i) = -\log P_X(a_i)$ der in Bemerkung 5.16 definierte Überraschungswert. Dann haben die unabhängigen Zufallsvariablen $h(X_1), \ldots, h(X_N)$ alle Erwartungswert $H(X)$ und Varianz σ^2 (deren genauer Wert uns nicht interessiert), so dass wir das Gesetz der großen Zahlen (Satz 5.5) auf $h(X_1), \ldots, h(X_N)$ anwenden können, wodurch wir (mit $\alpha = \beta^2$)
$$P\left(\left(\frac{1}{N}\sum_{i=1}^{N} h(X_i) - H(X)\right)^2 \geq \beta^2\right) \leq \frac{\sigma^2}{\beta^2 N} \tag{5.6}$$
erhalten. Nun gilt
$$\frac{1}{N}\sum_{i=1}^{N} h(X_i) = \frac{1}{N}\sum_{i=1}^{N} -\log P_X(X_i) = -\frac{1}{N}\log\prod_{i=1}^{N} P_X(X_i) = -\frac{1}{N}\log P_{X^N}(X^N)$$
und damit ist (5.6) äquivalent zu
$$P\left(\left|-\frac{1}{N}\log P_{X^N}(X^N) - H(X)\right| \geq \beta\right) \leq \frac{\sigma^2}{\beta^2 N}.$$
Daraus folgt dann
$$\begin{aligned} P(X^N \notin T_{N,\beta}) &= P\left(\left|-\frac{1}{N}\log P_{X^N}(X^N) - H(X)\right| > \beta\right) \\ &< P\left(\left|-\frac{1}{N}\log P_{X^N}(X^N) - H(X)\right| \geq \beta\right) \leq \frac{\sigma^2}{\beta^2 N} \end{aligned}$$

und über das Gegenereignis $P(X^N \in T_{N,\beta}) > 1 - \frac{\sigma^2}{\beta^2 N}$; mit $N_0 = \left\lceil \frac{\sigma^2}{\beta^2 \varepsilon} \right\rceil$ folgt die Behauptung.

2. Es gilt
$$1 \geq \sum_{x \in T_{N,\beta}} P_{X^N}(x) \geq \sum_{x \in T_{N,\beta}} 2^{-N(H(X)+\beta)} = \left|T_{N,\beta}\right| 2^{-N(H(X)+\beta)},$$
wobei die zweite Ungleichung aus (5.5) folgt, und somit $\left|T_{N,\beta}\right| \leq 2^{N(H(X)+\beta)}$. □

Bemerkung 5.53 Da es insgesamt $|A|^N$ mögliche N-Wörter gibt, erhalten wir als Quotient

$$\frac{\left|T_{N,\beta}\right|}{|A|^N} \leq \frac{2^{N(H(X)+\beta)}}{2^{N \log|A|}} = 2^{-N(\log|A| - H(X) - \beta)}.$$

Für $H(X) < \log|A|$ und β klein genug wird der Exponent negativ, so dass das Verhältnis von $\left|T_{N,\beta}\right|$ zu $|A|^N$ mit wachsendem N *exponentiell* abnimmt.

Ist andererseits $H(X) = \log|A|$, also nach Lemma 5.23 X gleichverteilt, sind alle N-Wörter gleich wahrscheinlich und $T_{N,\beta} = A^N$ für beliebiges $\beta > 0$. Die typische Menge ist also nur für nicht gleichverteilte Zufallsvariablen von Interesse. ◁

Zum Beweis von Satz 5.48 benötigen wir typische N-Wörter $(x, y) \in A^N \times B^N$ von zweidimensionalen Zufallsvariablen (X, Y) (das werden später Ein- und Ausgabe des Kanals sein), allerdings mit der etwas stärkeren Bedingung, dass nicht nur (x, y) typisch für P_{XY} ist, sondern darüber hinaus sollen die beiden Komponenten x und y auch typisch bezüglich der jeweiligen Marginalverteilung P_X und P_Y sein.

Definition 5.54 Sei (X, Y) eine Zufallsvariable mit Alphabet $A \times B$. Ein N-Wort $(x, y) \in A^N \times B^N$ heißt *gemeinsam β-typisch für P_{XY}*, wenn

1. x β-typisch für P_X ist, also $\left|-\frac{1}{N} \log P_{X^N}(x) - H(X)\right| \leq \beta$,
2. y β-typisch für P_Y ist, also $\left|-\frac{1}{N} \log P_{Y^N}(y) - H(Y)\right| \leq \beta$, und
3. (x, y) β-typisch für P_{XY} ist, also $\left|-\frac{1}{N} \log P_{(XY)^N}(x, y) - H(X, Y)\right| \leq \beta$.

Die Menge der gemeinsam β-typischen Wörter der Länge N bezeichnen wir mit

$$T'_{N,\beta}(X, Y) := \left\{(x, y) \in A^N \times B^N : x \in T_{N,\beta}(X) \text{ und } y \in T_{N,\beta}(Y) \text{ und } (x, y) \in T_{N,\beta}(X, Y)\right\}.$$
◁

Obwohl nach Definition $T'_{N,\beta}(X, Y) \subseteq T_{N,\beta}(X, Y)$ gilt, werden wir nun zeigen dass die beiden Aussagen von Lemma 5.52 auch für gemeinsam typische Wörter gelten. Außerdem benötigen wir noch eine dritte Eigenschaft; diese schätzt die Wahrscheinlichkeit nach oben ab, dass ein N-Wort (x, y) gemeinsam typisch ist, wenn dessen x- und y-Komponente *unabhängig voneinander* nach den Marginalverteilungen P_{X^N} und P_{Y^N} verteilt sind.

Lemma 5.55 *Seien eine 2-dimensionalen Zufallsvariable (X, Y), $N \in \mathbb{N}$ und $\beta > 0$ gegeben.*

1. *Zu jedem $\varepsilon > 0$ existiert ein $N_0 \in \mathbb{N}$, so dass für alle $N > N_0$ gilt:*

$$P\left((X,Y)^N \in T'_{N,\beta}(X,Y)\right) > 1 - \varepsilon.$$

2. $\left|T'_{N,\beta}\right| \leq 2^{N(H(X,Y)+\beta)}$.

3. *Sei (X', Y') eine Zufallsvariable auf $A \times B$ mit der Verteilung $P_{X'Y'}(a,b) = P_X(a)P_Y(b)$, also die unabhängige Kombination von X und Y (vgl. den Beweis von Lemma 5.26). Dann gilt*

$$P\left((X',Y')^N \in T'_{N,\beta}\right) \leq 2^{-N(I(X\|Y)-3\beta)}.$$ ◁

Beweis. Teil 1 und Teil 2 sind Übungsaufgaben.

Teil 3: Es gilt

$$\begin{aligned}
P\left((X',Y')^N \in T'_{N,\beta}\right) &= \sum_{(x,y)\in T'_{N,\beta}} P_{X^N}(x)P_{Y^N}(y) \\
&\leq \sum_{(x,y)\in T'_{N,\beta}} 2^{-N(H(X)-\beta)}2^{-N(H(Y)-\beta)} && \text{nach (5.5)} \\
&= \left|T'_{N,\beta}\right| 2^{-N(H(X)-\beta)}2^{-N(H(Y)-\beta)} \\
&\leq 2^{N(H(X,Y)+\beta)}2^{-N(H(X)-\beta)}2^{-N(H(Y)-\beta)} && \text{nach Teil 2} \\
&= 2^{-N(H(X)+H(Y)-H(X,Y)-3\beta)} \\
&= 2^{-N(I(X\|Y)-3\beta)},
\end{aligned}$$

wobei wir im ersten Schritt die Definition $P_{X'Y'^N}(x,y) = P_{X^N}(x)P_{X^N}(y)$ und im zweiten die Voraussetzungen $x \in T_{N,\beta}(X)$, $y \in T_{N,\beta}(y)$ genutzt haben. □

Beispiel 5.56 Sei (X, Y) der In- und Output eines BSC($\frac{1}{5}$) bei Eingangsverteilung $p_0 = P_X(0) = 0{,}9$, $p_1 = P_X(1) = 0{,}1$. Dann ist $P_Y(1) = \frac{9}{10} \cdot \frac{1}{5} + \frac{1}{10} \cdot \frac{4}{5} = \frac{13}{50} = 0{,}26$. Ist (x, y) ein gemeinsam typisches Wort der Länge $N = 100$, muss x ungefähr 10 und y ungefähr 26 Einser enthalten. Darüber hinaus muss aber auch (x, y) typisch für (X, Y) sein. Mit ein wenig Rechnerei kann man zeigen, dass dies gerade bedeutet, dass auch die Anzahl der Übertragungsfehler auf dem Kanal nah bei der erwarteten Anzahl $N\varepsilon = 20$ liegen. Beispielsweise sind (x, y) mit

```
x=1111111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
y=0011111111000000000000000000000000000000000000000000000000000000000000000000000011111111111111111
```

gemeinsam typisch. ◁

5.2.4. Beweis des Kanalcodierungssatzes (Satz 5.48)

Wir benötigen ein letztes kleines Lemma, das Sie in den Übungen beweisen werden.

Lemma 5.57 *Sei X eine Zufallsvariable mit Alphabet $A \subseteq \mathbb{R}$ und Erwartungswert $\mu_X = \mathbb{E}[X]$.*

1. *Es gibt mindestens ein $x \in A$ mit $x \leq \mu_X$.*
2. *Ist X gleichverteilt, $A \subseteq \mathbb{R}_0^+$ und $|A|$ gerade, gilt $x \leq 2\mu_X$ für mindestens die Hälfte aller $x \in A$.* ◁

Sei nun zum Beweis von Satz 5.48 ein Kanal $\mathcal{K} = (B, P_{\mathcal{K},0}, P_{\mathcal{K},1})$ mit Kapazität C, eine Rate $R < C$ sowie $\varepsilon > 0$ gegeben. Außerdem sei P_X die Verteilung auf dem Alphabet $\mathrm{GF}(2)$, welche in Definition 5.38 die wechselseitige Information maximiert und damit zur Kapazität C macht.

Wähle zunächst ein $R' \in (R, C) \cap \mathbb{Q}$ beliebig, ein positives $\beta < (C - R')/3$ und setze $\delta := \varepsilon/4$ sowie N „ausreichend groß“ (im Folgenden werden einige Aussagen für „große“ N gelten; das am Ende gewählte N muss dann mindestens das Maximum all dieser Mindestgrößen sein). Konstruiere mit diesen Daten folgendes Kommunikationssystem:

1. Wähle einen *zufälligen* (N, K)-Blockcode mit $K = NR'$, indem jedes der $2^K = 2^{NR'}$ Codewörter x nach der Verteilung $P_{X^N}(x) = \prod_{i=1}^N P_X(x_i)$ zufällig gewürfelt wird (mit dem P_X aus der Kanalkapazität). Die einzelnen Codewörter werden unabhängig voneinander gewählt, wodurch wir den *zufälligen Code* $\bar{\mathscr{C}}$ erhalten; dass dabei auch Codewörter mehrfach auftreten könnten, stört uns nicht weiter. (Damit es einen (N, K)-Blockcode gibt, muss $K = NR' \in \mathbb{N}$ sein. Falls die Bedingung für unser vorher bestimmtes „großes N“ nicht gilt, können wir es mit dem Nenner von $R' \in \mathbb{Q}$ multiplizieren, um ein noch größeres N mit $NR' \in \mathbb{N}$ zu erhalten)

2. Sender und Empfänger einigen sich auf diesen zufällig gewählten Code $\bar{\mathscr{C}}$ sowie eine Codierungsfunktion $E_{\bar{\mathscr{C}}} \colon \mathrm{GF}(2)^K \to \bar{\mathscr{C}}$ (deren genaue Form uns wieder nicht interessiert), bevor es zur eigentlichen Kommunikation kommt.

3. Der Sender erhält ein K-Wort aus der Quelle, codiert es zu einem Codewort x, und schickt dieses über den Kanal. Beim Empfänger kommt ein entsprechendes $y \in B^N$ an. Hier ist gleich dreimal der Zufall im Spiel: bei der Wahl von $\bar{\mathscr{C}}$, bei der Wahl des Codeworts $x \in \bar{\mathscr{C}}$, und bei der verrauschten Kanalausgabe y.

4. Der Empfänger decodiert y mittels eines *Typische-Menge-Decodierers*

$$D_{\bar{\mathscr{C}}}(y) := \begin{cases} x \in \bar{\mathscr{C}} & (x, y) \in T'_{N,\beta}(X, Y) \text{ und } (x', y) \notin T'_{N,\beta}(X, Y) \text{ für alle } x \neq x' \in \bar{\mathscr{C}}, \\ \mathsf{e} & \text{sonst.} \end{cases}$$

   $D_{\bar{\mathscr{C}}}$ entscheidet sich also genau dann für ein Codewort x, wenn es das einzige mit y gemeinsam β-typische Codewort aus $\bar{\mathscr{C}}$ ist.

Obwohl in dieser Konstruktion alles sehr suboptimal aussieht (der zufällige Code könnte zum Beispiel, im schlimmsten Fall, 2^K-mal das gleiche Codewort enthalten!), werden die starken Eigenschaften von $T'_{N,\beta}$ erlauben, den Satz zu beweisen.

Der Typische-Menge-Decodierer macht nach Definition einen Decodierfehler, wenn x nicht das einzige Codewort mit $(x, y) \in T'_{N,\beta}$ ist. Dazu unterscheiden wir die Ereignisse

F1: $(x, y) \notin T'_{N,\beta}$: Das gesendete x ist gar nicht gemeinsam typisch mit der Kanalausgabe,

F2: es gibt ein anderes $x' \in \bar{\mathscr{C}}$, $x' \neq x$, das mit y gemeinsam typisch ist: $(x', y) \in T'_{N,\beta}$.

F1 und F2 sind nicht disjunkt, decken aber sämtliche Bedingungen ab, unter denen es zum Decodierfehler kommt: Ist $D_{\bar{\mathscr{C}}}(y) = \mathtt{e}$, gibt es entweder gar kein mit y gemeinsam typisches Codewort (F1) oder mindestens zwei (F2). Ist $D_{\bar{\mathscr{C}}}(y) = x' \in \mathscr{C}$, $x' \neq x$, muss es genau ein $x' \neq x$ mit $x' \in T'_{N,\beta}$ geben (F1 und F2). Somit gilt

$$p_{\text{Err}}(x) = P(\text{F1 oder F2}) \leq P(\text{F1}) + P(\text{F2}). \tag{5.7}$$

Um zu beweisen, dass ein Code existiert, dessen maximaler Blockfehler $< \varepsilon$ ist, schätzen wir zunächst die *über alle zufälligen Codes* $\bar{\mathscr{C}}$ gemittelte durchschnittliche Blockfehlerrate

$$\bar{p}_{\text{Err}} := \frac{1}{|M|} \sum_{\mathscr{C} \in M} p_{\text{Err}}(\mathscr{C}) = \mathbb{E}\left[p_{\text{Err}}(\bar{\mathscr{C}})\right]$$

ab, wobei M die Menge aller (N, K)-Blockcodes ist. Dies klingt zwar kompliziert, ist aber viel einfacher, als $p_{\text{Err}}(\mathscr{C})$ für einen *bestimmten* Code auszurechnen: durch zufällige Wahl des Codes wird nämlich die Kanaleingabe zur Zufallsvariable X^N, so dass wir Lemma 5.55 anwenden können.

F1: Das gesendete Codewort des zufälligen Codes $\bar{\mathscr{C}}$ entspricht nach Konstruktion der Zufallsvariablen X^N mit Verteilung P_{X^N}, und wie in Bemerkung 5.36 erhalten wir die zweidimensionale Zufallsvariable $(X, Y)^N$ für Kanal-Ein- und -Ausgabe beim Verschicken dieses zufälligen Codeworts. Nach Lemma 5.55, Teil 1 ist $P(\text{F1}) = P\left((X, Y)^N \notin T'_{N,\beta}(X, Y)\right) < \delta$ für ausreichend großes N.

F2: Auch jedes andere, nicht dem gesendeten entsprechende zufällige Codewort ist wegen der zufälligen Konstruktion von $\bar{\mathscr{C}}$ eine Zufallsvariable $\tilde{X}^N$ mit Verteilung $P_{\tilde{X}^N} = P_{X^N}$, hat aber im Gegensatz zum tatsächlich gesendeten X^N nichts mit der Kanalausgabe Y^N zu tun, ist also unabhängig von Y^N, so dass

$$P_{(\tilde{X}Y)^N}(\tilde{x}, y) = \sum_{i=1}^{N} P_X(\tilde{x}_i) P_Y(y_i).$$

Somit können wir Teil 3 von Lemma 5.55 anwenden und erhalten

$$P\left((\tilde{X}^N, Y^N) \in T'_{N,\beta}\right) \leq 2^{-N(I(X\|Y)-3\beta)} = 2^{-N(C-3\beta)}$$

als Wahrscheinlichkeit für *ein* nicht gesendetes Codewort, gemeinsam typisch mit der Kanalausgabe zu sein. Da der Code insgesamt 2^K Codewörter enthält, gibt es $2^K - 1 =$

$2^{NR'} - 1$ solche nicht gesendeten Codewörter, die für $\tilde{X}^N$ infrage kommen, und wir erhalten

$$P(\text{F2}) \leq (2^{NR'} - 1) \cdot 2^{-N(C-3\beta)} \leq 2^{NR'} \cdot 2^{-N(C-3\beta)} = 2^{-N(C-R'-3\beta)}.$$

Insgesamt gilt damit für $N > N_0$ nach (5.7)

$$\bar{p}_{\text{Err}} = P(\text{F1 oder F2}) \leq P(\text{F1}) + P(\text{F2}) \leq \delta + 2^{-N(C-R'-3\beta)}.$$

Da $C - R' - 3\beta > C - R' - 3\frac{C-R'}{3} = 0$ ist der Exponent von $2^{-N(C-R'-3\beta)}$ negativ und damit $\bar{p}_{\text{Err}} = \mathbb{E}[p_{\text{Err}}(\bar{\mathscr{C}})] < 2\delta = \varepsilon/2$ für hinreichend große N. Nach Teil 1 von Lemma 5.57 muss dann aber auch für *mindestens einen Code* $\mathscr{C}'$ die Bedingung $p_{\text{Err}}(\mathscr{C}') < \varepsilon/2$ gelten.

Wir haben es fast geschafft. Allerdings wollen wir noch ein bisschen mehr, nämlich einen Code mit *maximaler* Fehlerwahrscheinlichkeit $p^*_{\text{Err}} < \varepsilon$. Die Aussage $p_{\text{Err}}(\mathscr{C}') < \varepsilon/2$ schließt noch nicht aus, dass einzelne Codewörter mit extrem hoher Wahrscheinlichkeit falsch decodiert werden (ist ein Codewort zum Beispiel nicht typisch für X, wird es nie richtig decodiert). Dem kommen wir bei, indem wir einfach die schlechteste Hälfte der Codewörter, also die $x \in \mathscr{C}'$, für die $p_{\text{Err}}(x)$ am größten ist, wegschmeißen. Damit erhalten wir einen Code $\mathscr{C}$, für den nach Teil 2 von Lemma 5.57 $p^*_{\text{Err}} < \varepsilon$ gilt. Da wir die Hälfte der Codewörter entfernt haben, gilt

$$|\mathscr{C}| = |\mathscr{C}'|/2 = 2^K/2 = 2^{K-1} = 2^{NR'-1}$$

und die Rate von $\mathscr{C}$ ist $\frac{K-1}{N} = R' - \frac{1}{N}$. Da $R' > R$ nach Wahl, ist für ausreichend große N auch $R' - \frac{1}{N} > R$, und $\mathscr{C}$ erfüllt alle Behauptungen von Satz 5.48.

6. Codierungstheorie

So schön die Aussagen von Satz 5.48 sind, bleiben bei der praktischen Umsetzung einige Fragen und Probleme offen:

- Die Aussage gilt für „sehr großes“ N, in der Praxis muss die Blocklänge aber beherrschbar groß bleiben, damit die Latenz nicht zu groß wird: der Codierer eines (N, K)-Blockcodes kann einen Block erst verschicken, wenn K Bits aus der Quelle gekommen sind. Das erste dieser K Bits ist dann schon K Takte „alt“, was z. B. beim Telefonieren zu unerwünschten Verzögerungen führen kann. Außerdem führen größere Blocklängen generell zu komplexeren Algorithmen beim Decodieren.
- Die zufällige Codeerzeugung im Beweis von Satz 5.48 ist nicht praktikabel. Erstens haben wir für einen gegebenen Code keine Möglichkeit herauszufinden, ob dieser wirklich gut genug oder zufällig schlecht ist. Außerdem ist es unmöglich, alle 2^K N-Wörter einzeln auszuwürfeln bzw. überhaupt nur aufzuschreiben. In der Praxis ist z. B. $(N, K) = (1024, 512)$ eine übliche Größe. Ein solcher Code hat 2^{512} Codewörter mit jeweils 1024 bit. Um diesen Code aufzuschreiben, bräuchten wir $2^{512} \cdot 1024 = 2^{522}$ Nullen und Einsen (zum Vergleich: unser Universum hat geschätzt ungefähr 2^{240} Atome …).
- Der im Beweis verwendete Typische-Menge-Decodierer ist aus den gleichen Gründen hoffnungslos ineffizient: er müsste für ein empfangenes $y \in B^N$ *jedes* der 2^K Codewörter $x \in \mathscr{C}$ durchgehen und ausrechnen ob $(x, y) \in T'_{N,\beta}$ gilt.

Deshalb werden Codes benötigt, deren Struktur man möglichst kompakt beschreiben und damit rechnen kann. In der Praxis werden heute fast ausschließlich *lineare Codes* benutzt, die man bequem mit Matrizen darstellen kann und die wir in Abschnitt 6.2 behandeln. Vorher untersuchen wir noch einige allgemeine Eigenschaften von Codes, die deren Fehlerkorrekturfähigkeit bestimmen.

Bemerkung 6.1 Natürlich waren unsere Bemühungen in Kapitel 5 trotz der mangelnden Praktikabilität nicht vergebens: immerhin wissen wir jetzt, bei welchen Codierungsraten wir überhaupt auf gute Fehlerkorrektur hoffen dürfen. Außerdem haben wir ein paar spannende Anschauungen und Resultate rund um die Entropie gewonnen und verstanden, was beim Zusammenfassen großer Datenblöcke passiert (Stichwort Typizität)! ◁

Bemerkung 6.2 In diesem Kapitel werden wir $\mathrm{GF}(2)^N$ als N-dimensionalen Vektorraum über dem Grundkörper $\mathrm{GF}(2) = \{0, 1\}$ auffassen anstatt wie bisher nur als N-Tupel von Bits. Dies erlaubt uns, Elemente von $\mathrm{GF}(2)^N$ zu addieren, und Konzepte wie lineare Abhängigkeit, Dimension etc. aus der linearen Algebra zu verwenden. Zu beachten ist, dass in $\mathrm{GF}(2)$ wegen $1 + 1 = 0$ die Gleichung $1 = -1$ gilt, weshalb in $\mathrm{GF}(2)^N$ Addition und Subtraktion identisch sind.

In der Codierungstheorie ist es üblich, Vektoren als *Zeilenvektoren*, also $1 \times N$-Matrizen zu definieren. Ist $w = w_1 \cdots w_N \in \mathrm{GF}(2)^N$, bezeichnet w^T den entsprechenden Spaltenvektor, also eine $N \times 1$-Matrix. ◁

6.1. ML-Decodierung und Codewort-Abstände

Bei der Übertragung eines Codeworts über den BSC werden einige Bits vertauscht (von 0 zu 1 oder umgekehrt). Wenn dadurch das gesendete Codewort x in ein anderes Codewort $x' \neq x$ übergeht, hat der Decodierer keine Chance, diesen Fehler zu erkennen – er wird davon ausgehen, dass x' gesendet und ohne Fehler übertragen wurde. Bei einem guten Code sollte also der „Abstand“ zwischen zwei Codewörtern möglichst groß sein. Dazu definieren wir:

Definition 6.3 (Hamming-Abstand) Sei $N \in \mathbb{N}$ und $x, y \in \mathrm{GF}(2)^N$. Dann heißt

$$\mathrm{d}(x, y) \coloneqq |\{i \colon x_i \neq y_i\}|$$

der *Hamming-Abstand* von x und y. ◁

Lemma 6.4 *Der Hamming-Abstand definiert eine* Metrik *auf* $\mathrm{GF}(2)^N$*, d. h. für* $u, v, w \in \mathrm{GF}(2)^N$ *gilt*

1. Positive Definitheit: $\mathrm{d}(u, v) \geq 0$ *und* $\mathrm{d}(u, v) = 0$ *genau für* $u = v$*,*
2. Symmetrie: $\mathrm{d}(u, v) = \mathrm{d}(v, u)$
3. Dreiecksungleichung: $\mathrm{d}(u, v) \leq \mathrm{d}(u, w) + \mathrm{d}(w, v)$

Außerdem ist er translationsinvariant*:*

4. $\mathrm{d}(u + w, v + w) = \mathrm{d}(u, v)$. ◁

Beweis. Übungsaufgabe. □

Der im Beweis von Satz 5.48 verwendete Typische-Menge-Decodierer ist nur für theoretische Zwecke interessant und in der Praxis weder realisierbar noch unbedingt sonderlich gut. Wir definieren nun stattdessen die *Maximum-Likelihood-Decodierung*, die unter bestimmten Voraussetzungen *optimal* für gegebenen Code und Kanal ist, also die kleinstmögliche Fehlerwahrscheinlichkeit erreicht.

Definition 6.5 Sei $\mathscr{C}$ ein (N, K)-Blockcode und $\mathscr{K}$ ein Kanal mit Ausgabealphabet B. Für $y \in B^N$ und $x \in \mathscr{C}$ bezeichne $P(y \mid x) \coloneqq \prod_{i=1}^{N} P_{\mathscr{K}, x_i}(y_i)$ die Wahrscheinlichkeit, dass y beim Empfänger ankommt, wenn x gesendet wird. Ein Decodierer D_{ML}, der für alle $y \in B^N$

$$D_{\mathrm{ML}}(y) = \operatorname*{arg\,max}_{x \in \mathscr{C}} P(y \mid x)$$

erfüllt, heißt *Maximum-Likelihood (ML)-Decodierer*. Ein (nicht notwendigerweise eindeutiges) Codewort $\hat{x}$ mit $P(y \mid \hat{x}) = \max_{x \in \mathscr{C}} P(y \mid x)$ heißt *ML-Codewort* für y. ◁

Bemerkung 6.6 Fasst man einen Ausdruck der Form $P(y \mid x)$ als *Funktion von x*, also der Bedingung auf, nennt man dies in der Statistik eine *Likelihood-Funktion*. Wird eine Entscheidung basierend auf dem Maximum dieser Funktion getroffen, spricht man vom *Maximum-Likelihood-Schätzer*.

Der ML-Decodierer entscheidet sich für das Codewort x, welches unter allen am wahrscheinlichsten den Output y hervorruft. In den Übungen werden Sie zeigen, dass dies die optimale Strategie ist, wenn der Sender alle Codewörter mit gleicher Wahrscheinlichkeit verschickt.

Leider ist die ML-Decodierung für fast alle Codes von praktischer Relevanz sehr schwer: es gibt keinen Algorithmus, der in vertretbarer Zeit ein ML-Codewort findet. Allerdings gibt es zahlreiche Decodieralgorithmen, die für spezielle Klassen von Codes zumindest recht nah an die Fehlerkorrekturrate eines ML-Decodierers kommen, diesen also *approximieren*. ◁

Wir zeigen nun, dass beim BSC(ε) ML-Codewörter gerade die mit kleinstem Hamming-Abstand zur Kanalausgabe y sind.

Lemma 6.7 *Bei der Nachrichtenübertragung eines (N, K)-Blockcode $\mathscr{C}$ über den BSC(ε) werde $x \in \mathscr{C}$ gesendet und $y \in \mathrm{GF}(2)^N$ empfangen. Dann gilt für $\hat{x} := D_{\mathrm{ML}}(y)$:*

$$\mathrm{d}(\hat{x}, y) = \min_{x \in \mathscr{C}} \mathrm{d}(x, y),$$

ein ML-Codewort hat also unter allen Codewörtern den kleinsten Hamming-Abstand zu y, und ein Decodierer, der zu jedem $y \in \mathrm{GF}(2)^N$ ein $x \in \mathscr{C}$ mit kleinstem Abstand findet, ist ein ML-Decodierer. ◁

Beweis. Für $\mathrm{d}(x, y) = l$ (l Bits falsch und damit $N - l$ korrekt übertragen) gilt

$$P(y \mid x) = \prod_{i=1}^{N} P_{\mathrm{BSC}(\varepsilon), x_i}(y_i) = \varepsilon^l (1-\varepsilon)^{N-l} = \frac{\varepsilon^l (1-\varepsilon)^N}{(1-\varepsilon)^l} = \left(\frac{\varepsilon}{1-\varepsilon}\right)^l (1-\varepsilon)^N.$$

Wegen $\varepsilon < \frac{1}{2}$ ist $\frac{\varepsilon}{1-\varepsilon} < 1$, so dass $P(y \mid x)$ mit wachsendem l streng monoton fällt, oder anders gesagt für kleinere l größer ist. Um $P(y \mid x)$ zu maximieren muss ein ML-Codewort $\hat{x}$ also l minimieren: $\mathrm{d}(\hat{x}, y) = \min_{x \in \mathscr{C}} \mathrm{d}(x, y)$. □

Zur Vermeidung von Fehlern ist es nach Lemma 6.7 wünschenswert, dass zwischen je zwei Codewörtern möglichst viel Platz ist. Dazu definieren wir

Definition 6.8 Sei $\mathscr{C}$ ein (N, K)-Blockcode. Wir nennen

$$\mathrm{d}(\mathscr{C}) := \min\{\mathrm{d}(x, x') : x, x' \in \mathscr{C}, x \neq x'\}$$

die *Minimaldistanz* von $\mathscr{C}$. Für den Fall $|\mathscr{C}| = 1$ legen wir $\mathrm{d}(\mathscr{C}) = 0$ fest. ◁

Für $x \in \mathbb{R}^3$ ist die Kugel um x mit Radius $r > 0$ definiert als die Menge aller Punkte y, die von x höchstens den Abstand $\|x - y\|_2 \leq r$ haben. Mit dem Hamming-Abstand können wir äquivalent „Kugeln“ im $\mathrm{GF}(2)^N$ definieren.

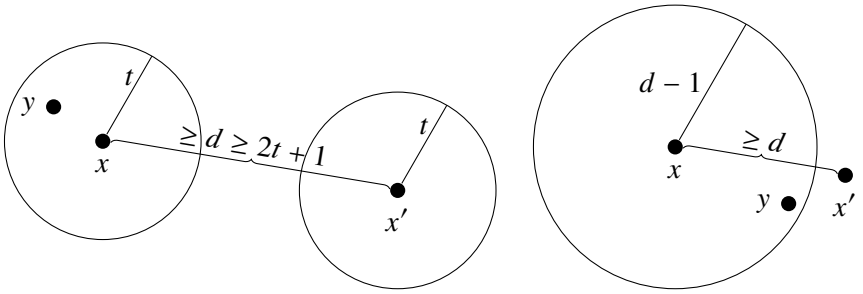


Abbildung 6.1.: Links: disjunkte t-Kugeln um verschiedene Codewörter eines Codes $\mathscr{C}$ mit Minimaldistanz $\mathrm{d}(\mathscr{C}) \geq 2t+1$. Fällt y in $\mathrm{B}_t(x)$, kann es eindeutig decodiert werden. Rechts: Da $\mathrm{B}_{d-1}(x)$ außer x keine Codewörter erkennt, wird ein Fehler erkannt, wenn y in diese Kugel fällt.

Definition 6.9 Für $x \in \mathrm{GF}(2)^N$ und $r \in \mathbb{N}_0$ heißt

$$\mathrm{B}_r(x) = \{y \in \mathrm{GF}(2)^N \colon \mathrm{d}(x,y) \leq r\}$$

die *Kugel* vom Radius r um den Mittelpunkt $x \in \mathrm{GF}(2)^N$. ◁

Bemerkung 6.10 Für $x \in \mathrm{GF}(2)^N$ und $j \in \mathbb{N}_0$ ist $|\{y \colon \mathrm{d}(x,y) = j\}| = \binom{N}{j}$ (wähle j der N Einträge von x aus, die geändert werden). Somit ist

$$|\mathrm{B}_r(x)| = \sum_{j=0}^{r} \binom{N}{j}.$$

◁

Satz 6.11 *Sei $\mathscr{C}$ ein Blockcode, $d \coloneqq \mathrm{d}(\mathscr{C})$ und $t \coloneqq \left\lfloor \frac{d-1}{2} \right\rfloor$. Wird $x \in \mathscr{C}$ über den BSC(ε) übertragen, $y \in \mathrm{GF}(2)^N$ empfangen und gilt $\mathrm{d}(x,y) \leq t$, folgt $D_{\mathrm{ML}}(y) = x$: die ML-Decodierung ist erfolgreich.* ◁

Beweis. Aus $t = \lfloor \frac{d-1}{2} \rfloor$ folgt $d \geq 2t+1$. Dann gilt

$$\mathrm{B}_t(x) \cap \mathrm{B}_t(x') = \emptyset \tag{6.1}$$

für alle $x, x' \in \mathscr{C}$ mit $x \neq x'$: gäbe es nämlich ein $v \in \mathrm{B}_t(x) \cap \mathrm{B}_t(x')$, erhalten wir mit der Dreiecksungleichung (Lemma 6.4, Punkt 3) $2t+1 \leq d \leq \mathrm{d}(x,x') \leq \mathrm{d}(x,v) + \mathrm{d}(v,x') \leq t+t = 2t$, ein Widerspruch (siehe Abbildung 6.1 links).

Gilt nun $\mathrm{d}(x,y) \leq t$, kommt es also bei der Übertragung von x zu höchstens t Fehlern, gilt $y \in \mathrm{B}_t(x)$ und damit $y \notin \mathrm{B}_t(x')$ für alle $x \neq x' \in \mathscr{C}$. Damit ist x das eindeutige Codewort mit $d(x,y) = \min_{x' \in \mathscr{C}} \mathrm{d}(x',y)$, also nach Lemma 6.7 das eindeutige ML-Codewort, für das sich D_{ML} entscheiden muss. □

Bei mehr als t Fehlern kann es passieren, dass der ML-Decodierer ein falsches Codewort ausgibt. Allerdings können wir eventuell immer noch *erkennen*, dass es zu Übertragungsfehlern

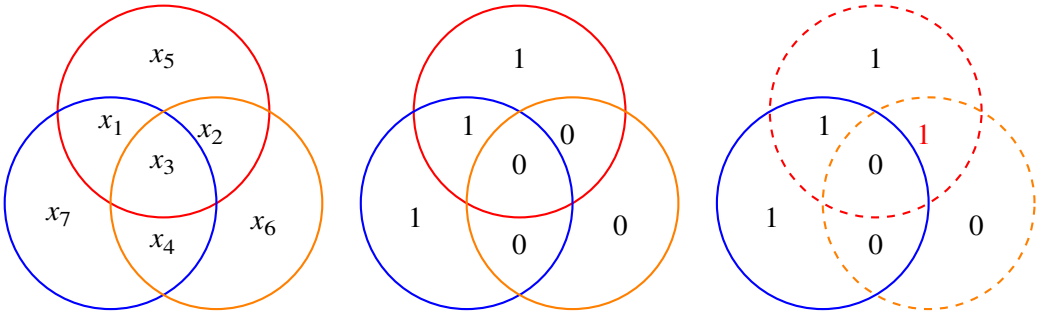


Abbildung 6.2.: *Links:* Schematische Darstellung des $(7,4)$-Hamming-Codes aus Beispiel 6.15. Die drei Kreise enstprechen den Teilmengen der Codewort-Einträge, die sich für die drei Bedingungen in der Definition des Codes zu 0 aufsummieren müssen, in denen es also jeweils eine gerade Anzahl Einser geben muss. *Mitte:* Beispielbelegung für das Codewort $x = 1000101$. *Rechts:* Bei einem Übertragungsfehler (hier in x_2) sind haben manche der Teilmengen (hier rot und orange) ungerade Parität. Es gibt genau eine Möglichkeit, durch Ändern eines einzigen Eintrags alle Paritäten wieder herzustellen.

gekommen ist: Ist $\mathrm{d}(x,y) \leq d-1$, so weiß der Decodierer, dass Fehler passiert sind, denn die Kugel $\mathrm{B}_{d-1}(x)$ enthält außer x kein weiteres Codewort $x' \in \mathscr{C}$ (Abbildung 6.1 rechts). Man nennt einen Code mit Minimaldistanz d deshalb auch $\lfloor (d-1)/2 \rfloor$-*fehlerkorrigierend* und $(d-1)$-*fehlererkennend*.

Beispiel 6.12 1. Der N-Wiederholungscode $R_N = \{00\cdots0, 11\cdots1\} \subseteq \mathrm{GF}(2)^N$ hat die Minimaldistanz $\mathrm{d}(R_N) = N$ (es gibt nur 2 Codewörter). Er kann somit (wie schon in Beispiel 4.33 besprochen) $\frac{N-1}{2}$ Fehler korrigieren und $N-1$ erkennen.

2. Der *Kontrollcode* auf $\mathrm{GF}(2)^N$ ist definiert als

$$\mathscr{C} := \left\{ x = x_1 \cdots x_N \in \mathrm{GF}(2)^N : \sum_{i=1}^{N} x_i = 0 \right\},$$

also die Menge aller N-Wörter mit sogenannter *gerader Parität* (gerade Anzahl Einser). Seine Minimaldistanz ist 2, so dass er keine Fehler korrigieren, aber einen erkennen kann.

3. Der $(7,4)$-*Hamming-Code*

$$\mathscr{C} := \left\{ (x_1,\dots,x_7) \in \mathrm{GF}(2)^7 : \begin{array}{r} x_1 + x_2 + x_3 + x_5 = 0 \\ x_2 + x_3 + x_4 + x_6 = 0 \\ x_1 + x_3 + x_4 + x_7 = 0 \end{array} \right\}$$

mit $N = 7$, der in Abbildung 6.2 dargestellt ist, hat genau 2^4 Codewörter: man kann nämlich jede der 2^4 Belegungen von $x_1,\dots,x_4$ frei wählen, woraufhin dann die drei Gleichungen die Werte von x_5, x_6, x_7 festlegen; somit haben wir einen $(7,4)$-Code.

Dieser Code kann mit folgendem Verfahren einen Fehler korrigieren: berechne für jede der drei Teilmengen $I_1 = \{1,2,3,5\}$, $I_2 = \{2,3,4,6\}$, $I_3 = \{1,3,4,7\}$, die Parität $\sum_{i\in I_j} x_i$. Bei einem Übertragungsfehler muss mindestens eine Parität ungerade sein; je nachdem, wo der Fehler passiert ist, entweder eine (Fehler in $\{x_5,x_6,x_7\}$), zwei ($\{x_1,x_2,x_4\}$) oder alle drei (Fehler in x_3) der Summen = 1. In jedem Fall gibt es genau eine Möglichkeit, durch Veränderung eines einzelnen Bits alle Paritäten wieder auf 0 zu setzen. Also können wir einen Fehler korrigieren und erhalten $\mathrm{d}(\mathscr{C}) \geq 3$. Andererseits ist wegen $0000000, 1110000 \in \mathscr{C}$ auch $\mathrm{d}(\mathscr{C}) \leq 3$, insgesamt also $\mathrm{d}(\mathscr{C}) = 3$. ◁

Durch die disjunkten t-Kugeln wissen wir, das bei ML-Decodierung im Fall von höchstens t Übertragungsfehlern korrekt decodiert wird. Bei mehr als t Fehlern hängt dies jedoch von der genauen Struktur des Codes ab: entweder landet y in der t-Kugel um ein falsches Codewort x' (dann kommt es zum Decodierfehler) oder aber im Zwischenraum zwischen allen t-Kugeln – in diesem Fall könnte x immer noch das Codewort mit kleinstem Abstand zu y sein, muss aber nicht. Da dieser Fall schwer zu analysieren ist, wäre es schön, wenn es gar keinen solchen Zwischenraum gäbe – dann nennt man einen Code *perfekt*.

Definition 6.13 Ein Code $\mathscr{C}$ mit Minimaldistanz $d \geq 1$ und $t = \lfloor (d-1)/2 \rfloor$ heißt *perfekt*, wenn

$$\mathrm{GF}(2)^N = \bigcup_{x\in\mathscr{C}} \mathrm{B}_t(x)$$

gilt, die (disjunkte) Vereinigung aller t-Kugeln um die Codewörter also den ganzen Raum ausfüllt. ◁

Wir haben oben gezeigt dass die t-Kugeln um die Codewörter disjunkt sind, also überlappungsfrei im Raum $\mathrm{GF}(2)^N$ liegen. Da dort aber nur endlich viel Platz ist, können die Kugeln nicht beliebig groß werden, und es kann nicht beliebig viele geben. Die folgende Ungleichung präzisiert diesen Zusammenhang.

Satz 6.14 (Kugelpackungsschranke) *Sei $\mathscr{C}$ ein (N,K)-Blockcode, $d = \mathrm{d}(\mathscr{C}) \geq 1$ und $t = \lfloor (d-1)/2 \rfloor$. Dann gilt die* Kugelpackungsschranke *(auch* Hamming-Schranke *genannt)*

$$2^K \sum_{i=0}^{t} \binom{N}{i} \leq 2^N,$$

die genau dann mit Gleichheit erfüllt ist, wenn $\mathscr{C}$ perfekt ist. ◁

Beweis. Nach (6.1) ist $\mathrm{B}_t(x) \cap \mathrm{B}_t(x') = \emptyset$ für alle Codeworte $x \neq x'$. Damit gilt

$$2^N = \left|\mathrm{GF}(2)^N\right| \geq \left|\bigcup_{x\in\mathscr{C}} \mathrm{B}_t(x)\right| = \sum_{x\in\mathscr{C}} |\mathrm{B}_t(x)| = \sum_{x\in\mathscr{C}} \sum_{i=0}^{t} \binom{N}{i} = |\mathscr{C}| \sum_{i=0}^{t} \binom{N}{i} = 2^K \sum_{i=0}^{t} \binom{N}{i}.$$

Die Gleichheit gilt nach Definition 6.13 genau für perfekte Codes. □

Beispiel 6.15 1. Der 3-Wiederholungscode aus Beispiel 4.33 und allgemein jeder N-Wiederholungscode mit ungeradem $N = 2t+1$ sind perfekt: es gibt nur die beiden Codewörter $0\cdots0$ und $1\cdots1$, und $\mathrm{B}_t(0\cdots0)$ enthält alle N-Wörter mit mehr Einträgen 0 als 1, während $\mathrm{B}_t(1\cdots1)$ genau den Rest enthält.

2. Für jedes N ist der triviale Code $\mathscr{C} := \mathrm{GF}(2)^N$ perfekt: seine Minimaldistanz ist $\mathrm{d}(\mathscr{C}) = 1$, also $t = 0$ und die 2^N „0-Kugeln“ enthalten jeweils nur das Codewort selbst, überdecken also ganz $\mathrm{GF}(2)^N$.

3. Der $(7,4)$-Hamming-Code aus Beispiel 6.12 ist nach Satz 6.14 perfekt, denn mit $t = \frac{3-1}{2} = 1$ gilt
$$2^4 \sum_{i=0}^{t} \binom{7}{i} = 16(1+7) = 16\cdot 8 = 128 = 2^7.$$
◁

Wir werden im nächsten Abschnitt noch eine Klasse perfekter Hamming-Codes mit Minimaldistanz $d = 3$ kennenlernen. Wir könnten nun darauf hoffen, die Aussage des Kanalcodierungssatzes (zumindest auf dem $\mathrm{BSC}(\varepsilon)$) mit perfekten Codes zu realisieren: wähle einen perfekten Code heraus, dessen Minimaldistanz d groß genug ist, dass die Wahrscheinlichkeit von mehr als $t = \lfloor (d-1)/2 \rfloor$ Übertragungsfehlern pro Codewort die gegebene Toleranz unterschreitet. Ein ML-Decodierer kann jedes empfangene N-Wort eindeutig decodieren, da es keinen Zwischenraum zwischen den t-Kugeln gibt. Leider ist dies nicht möglich:

Satz 6.16 *Außer den trivialen Codes aus Beispiel 6.15 mit $t \in \{0, (N-1)/2\}$, den Hamming-Codes mit $t = 1$ und einem $(23,12)$-Code mit $t = 3$ existieren keine weiteren perfekten Codes.* ◁

Den Beweis dieses Satzes lassen wir aus. Anschaulich liegt das Problem darin, dass bei den meisten Codes der Zwischenraum zwischen den t-Kugeln den meisten Raum ausfüllt. Es ist einfach nicht möglich, diese Kugeln so anzuordnen, dass dazwischen kein Platz frei bleibt.

Neben der Kugelpackungsschranke gibt es eine weitere Ungleichung, die N, K und d eines Blockcodes in Beziehung setzt.

Satz 6.17 (Singleton-Schranke) *Sei $\mathscr{C}$ ein (N,K)-Blockcode mit Minimaldistanz $\mathrm{d}(\mathscr{C}) = d$. Dann gilt die* Singleton-Schranke
$$d \le N - K + 1.$$
Erfüllt $\mathscr{C}$ diese Schranke mit Gleichheit, nennt man $\mathscr{C}$ einen MDS-Code *(*maximum distance separable*).* ◁

Beweis. Sei
$$\pi\colon \mathrm{GF}(2)^N \to \mathrm{GF}(2)^{N-d+1}$$
$$\pi(x_1\cdots x_N) = x_1\cdots x_{N-d+1}$$
die Projektion auf die ersten $N-d+1$ Komponenten. Da zwei verschiedene Codewörter von $\mathscr{C}$ sich in mindestens d Stellen unterscheiden, können die ersten $N-d+1$ Stellen nicht gleich sein, so dass π auf $\mathscr{C}$ injektiv ist und somit
$$2^K = |\mathscr{C}| = |\pi(\mathscr{C})| \le \left|\mathrm{GF}(2)^{N-d+1}\right| = 2^{N-d+1}$$

oder (nach Anwendung von log auf beiden Seiten) $K \leq N - d + 1$ bzw. $d \leq N - K + 1$ gilt. □

Der Begriff *maximum distance separable* erklärt sich aus dem Beweis: bei MDS-Codes können wir die Projektion auf beliebige $K = N - d + 1$ Koordinaten wählen und immer noch die verschiedenen Codewörter unterscheiden („separieren“). Außer trivialen Beispielen gibt es keine binären MDS-Codes. Wir werden später aber mit den *Reed-Solomon-Codes* eine Klasse von MDS-Codes über den Körpern $\mathrm{GF}(2^m)$ kennenlernen.

6.2. Lineare Codes

Im letzten Abschnitt haben wir uns mit den Abständen zwischen Codewörtern beschäftigt. Immer noch ungelöst ist allerdings das Problem, dass Codes praktischer Länge viel zu groß sein müssen, um alle Codewörter explizit aufzuschreiben. Der Ausweg besteht in *linearen Codes*, die in so gut wie allen praktischen Anwendungen vorkommen; einer ihrer Vorteile ist, dass sie sich kompakt durch Matrizen darstellen lassen.

Obwohl wir mit linearen Blockcodes nur noch eine kleine Teilmenge *aller* möglicher Codes betrachten, lässt sich zeigen, dass der Kanalcodierungssatz (Satz 5.48) auch für lineare Codes gilt; wir laufen also keine Gefahr, die „guten“ Codes durch diese Einschränkung zu übersehen.

Erinnerung 6.18 Sei V ein Vektorraum über dem Körper K.

1. Eine Teilmenge $U \subseteq V$ heißt *Untervektorraum* oder einfach *Unterraum*, wenn U selbst ein Vektorraum ist.
2. $U \subseteq V$ ist genau dann Unterraum, wenn das *Unterraumkriterium* erfüllt ist, d. h. für alle Vektoren $u, v \in U$ und alle Skalare $\alpha \in K$ gilt
   - $U \neq \emptyset$,
   - $u + v \in U$,
   - $\alpha u \in U$.
3. Eine Teilmenge $v_1, \ldots, v_l \subseteq V$ heißt *linear unabhängig*, wenn die Gleichung $\sum_{i=1}^{l} \lambda_i v_i = 0$ mit $\lambda_i \in K$ nur die eine Lösung $\lambda_1 = \cdots \lambda_l = 0$ hat.
4. Eine Teilmenge $B = \{b_1, \ldots, b_l\} \subseteq V$ heißt *Basis* von V, wenn B linear unabhängig ist und $\left\{\sum_{i=1}^{l} \lambda_i b_i \colon \lambda_i \in K\right\} = V$ ist, B also den ganzen Vektorraum V aufspannt. ◁

Korollar 6.19 *Sei V ein Vektorraum über dem Körper $\mathrm{GF}(2)$ und $0 \neq U \subseteq V$. U ist genau dann ein Unterraum von V, wenn $u + v \in U$ für alle $u, v \in U$ gilt.* ◁

Beweis. Da es in $\mathrm{GF}(2)^N$ nur die zwei Skalare 0 und 1 gibt, ist Bedingung 3 des Unterraumkriteriums automatisch erfüllt: für alle u gilt $0 \cdot u = 2 \cdot u = u + u \in U$ nach Voraussetzung und $1 \cdot u = u \in U$ sowieso. □

Definition 6.20 Ein (N, K)-Blockcode $\mathscr{C}$ heißt *linear*, wenn $\mathscr{C} \subseteq \mathrm{GF}(2)^N$ ein Untervektorraum der Dimension K ist.

Eine Matrix $G \in \mathrm{GF}(2)^{K\times N}$, deren Zeilen eine Basis von $\mathscr{C}$ bilden, heißt *Generatormatrix* für $\mathscr{C}$. ◁

Bemerkung 6.21 Ist $\mathscr{C}$ ein linearer (N, K)-Code und G eine Generatormatrix von $\mathscr{C}$, gilt

$$\mathscr{C} = \{w^T G \colon w \in \mathrm{GF}(2)^K\}.$$

Damit können wir G zum Codieren benutzen: die Abbildung

$$\begin{aligned} E_{\mathscr{C}} \colon \mathrm{GF}(2)^K &\to \mathrm{GF}(2)^N \\ w &\to wG \end{aligned}$$

ist nach Voraussetzung (Zeilen von G sind Basis, also linear unabhängig) injektiv und ihr Bild ist gerade der Unterraum $\mathscr{C} \subseteq \mathrm{GF}(2)^N$. Ein linearer (N, K)-Blockcode ist also durch die Angabe einer binären $K \times N$-Matrix eindeutig bestimmt und kann mit $K \cdot N$ Bits abgespeichert werden. Im Vergleich zu den $2^K \cdot N$ Bits für allgemeine Codes (vgl. Seite 117) ist dies eine drastische Reduktion; ein linearer $(1024, 512)$-Code bräuchte so ungefähr 512 KiB. ◁

Beispiel 6.22 Der $(7, 4)$-Hamming-Code aus Beispiel 6.12 ist linear: für die drei dort definierten Teilmengen I_1, I_2, I_3 und $x^1, x^2 \in \mathscr{C}$ gilt nach Definition $\sum_{i\in I_j} x_i^k = 0$ für $k = 1, 2$ und $j = 1, 2, 3$ und damit auch $\sum_{i\in I_j}(x^1 + x^2)_i = \sum_{i\in I_j} x_i^1 + \sum_{i\in I_j} x_i^2 = 0 + 0 = 0$ für $j = 1, 2, 3$, so dass $x^1 + x^2 \in \mathscr{C}$ und $\mathscr{C}$ damit nach Korollar 6.19 ein Unterraum von $\mathrm{GF}(2)^7$ ist.

Wie in Beispiel 6.12 bemerkt, können die ersten vier Einträge jedes Codeworts frei gewählt werden, wodurch dann x_5, x_6, x_7 bestimmt sind. Wählen wir die vier Einheitsvektoren, erhalten wir eine Basis von $\mathscr{C}$ und damit die Generatormatrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1. \end{pmatrix}$$

◁

Definition 6.23 Sei $N \in \mathbb{N}$. Für $x \in \mathrm{GF}(2)^N$ heißt

$$\mathrm{w}(x) \coloneqq \mathrm{d}(x, 0) = |\{i \colon x_i \neq 0\}|$$

das *Gewicht* von x. Für einen Code $\mathscr{C}$ heißt

$$\mathrm{w}(\mathscr{C}) \coloneqq \min_{0\neq x\in\mathscr{C}} \{\mathrm{w}(x)\}$$

das *Minimalgewicht* von $\mathscr{C}$. Für den Fall $\mathscr{C} \setminus \{0\} = \emptyset$ legen wir $\mathrm{w}(\mathscr{C}) = 0$ fest. ◁

Lemma 6.24 *Sei $\mathscr{C}$ ein linearer (N, K)-Code. Dann gilt $\mathrm{d}(\mathscr{C}) = \mathrm{w}(\mathscr{C})$: die Minimaldistanz entspricht dem minimalen Gewicht eines Codeworts.* ◁

Beweis. Für $|\mathscr{C}| = 1$ muss $\mathscr{C} = \{0\}$ gelten, da $\mathscr{C}$ als Vektorraum die Null enthalten muss. Also ist $\mathrm{w}(\mathscr{C}) = 0$. Nach Definition ist für $|\mathscr{C}| = 1$ auch $\mathrm{d}(\mathscr{C}) = 0$, womit die Aussage gilt.

Sei also $|\mathscr{C}| > 1$ und $x, x' \in \mathscr{C}$ mit $\mathrm{d}(x, x') = \mathrm{d}(\mathscr{C})$. Dann gilt

$$\mathrm{d}(x, x') = \mathrm{d}(x - x', x' - x') = \mathrm{d}(x - x', 0) = \mathrm{w}(x - x')$$

nach Lemma 6.4, Punkt 4. Wegen der Linearität von $\mathscr{C}$ ist $x - x' \in \mathscr{C}$, also $\mathrm{w}(\mathscr{C}) \leq \mathrm{w}(x - x') = \mathrm{d}(\mathscr{C})$.

Sei umgekehrt $x'' \in \mathscr{C}$ mit $\mathrm{w}(x'') = \mathrm{w}(\mathscr{C})$. Da $0 \in \mathscr{C}$ gilt $\mathrm{w}(x'') = \mathrm{d}(x'', 0) \geq \mathrm{d}(\mathscr{C})$ und damit insgesamt $\mathrm{w}(\mathscr{C}) = \mathrm{d}(\mathscr{C})$. □

Bemerkung 6.25 Auch wenn Lemma 6.24 die Bestimmung der Minimaldistanz etwas vereinfacht – statt $\mathrm{d}(x, y)$ für alle $\binom{|C|}{2}$ Paare von zwei Codewörtern zu berechnen, müssen wir „nur“ $\mathrm{w}(x)$ für alle $|\mathscr{C}| = 2^K$ Codewörter berechnen – ist dies für die meisten Codes nicht praktikabel. Abgesehen von dem Fall, dass sich die Minimaldistanz eines Codes anhand seiner speziellen Struktur beweisen lässt (wie bei den Wiederholungscodes), gibt es keine effiziente Methode, $\mathrm{d}(\mathscr{C})$ für einen beliebigen gegebenen (linearen) Code zu berechnen. ◁

Wir führen nun neben der Generatormatrix eine weitere Möglichkeit ein, einen linearen Code durch eine Matrix zu definieren.

Definition 6.26 Sei $\mathscr{C} \subseteq \mathrm{GF}(2)^N$ ein linearer (N, K)-Code. Eine Matrix $H \in \mathrm{GF}(2)^{(N-K)\times N}$ heißt *Kontrollmatrix* für $\mathscr{C}$, wenn

$$\mathscr{C} = \{x \in \mathrm{GF}(2)^N : Hx^T = 0\}$$

gilt. ◁

Beispiel 6.27 Die Kontrollmatrix des $(7, 4)$-Hamming-Codes ergibt sich direkt aus dessen Definition, die ja gerade fordert, dass sich bestimmte Einträge eines Codeworts zu 0 aufaddieren müssen: mit

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

haben wir

$$Hx^T = 0 \Leftrightarrow \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} x_1 + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} x_2 + \cdots + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} x_7 = 0 \Leftrightarrow \begin{cases} x_1 + x_2 + x_3 + x_5 = 0 \\ x_2 + x_3 + x_4 + x_6 = 0 \\ x_1 + x_3 + x_4 + x_7 = 0 \end{cases} \Leftrightarrow x \in \mathscr{C}.$$

◁

Lemma 6.28 *Jeder lineare (N, K)-Code $\mathscr{C}$ besitzt eine Kontrollmatrix H.* ◁

Beweis. Für $x, y \in \mathrm{GF}(2)^N$ bezeichne $\langle x, y\rangle := \sum_{i=1}^N x_i y_i$ das gewöhnliche Skalarprodukt. Dann ist das orthogonale Komplement zu $\mathscr{C}$,

$$\mathscr{C}^\perp := \{y \in \mathrm{GF}(2)^N : \langle x, y\rangle = 0 \text{ für alle } x \in \mathscr{C}\},$$

selbst ein Unterraum (und damit ein linearer Code), da offensichtlich $0 \in \mathscr{C}^\perp$ und für $y, y' \in \mathscr{C}^\perp$ gilt $\langle x, y + y'\rangle = \langle x, y\rangle + \langle x, y'\rangle = 0 + 0 = 0$, also $y + y' \in \mathscr{C}^\perp$.

Sei H eine Generatormatrix für $\mathscr{C}^\perp$ mit Zeilen $h_1, \dots, h_m$ mit $m = \dim(\mathscr{C}^\perp)$. Wir zeigen, dass H eine Kontrollmatrix für $\mathscr{C}$ ist, also dass $x \in \mathscr{C} \Leftrightarrow Hx^T = 0$ gilt.

„⇒“: Ist $x \in \mathscr{C}$, gilt

$$Hx^T = \begin{pmatrix} \langle h_1, x\rangle \\ \vdots \\ \langle h_m, x\rangle \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix},$$

da $h_i \in \mathscr{C}^\perp$ nach Definition.

„⇐“: Ist umgekehrt $Hx^T = 0$, also $\langle h_i, x\rangle = 0$ für alle $i = 1, \dots, m$. Jedes $y \in \mathscr{C}^\perp$ kann, da $h_1, \dots, h_m$ eine Basis von $\mathscr{C}^\perp$ ist, als $y = \sum_{i=1}^m \lambda_i h_i$ mit $\lambda_i \in \mathrm{GF}(2)$ geschrieben werden. Damit ist

$$\langle y, x\rangle = \left\langle \sum_{i=1}^m \lambda_i h_i, x \right\rangle = \sum_{i=1}^m \lambda_i \langle h_i, x\rangle = 0$$

für alle $y \in \mathscr{C}^\perp$, also $x \in (\mathscr{C}^\perp)^\perp = \mathscr{C}$ (die letzte Äquivalenz ist eine Übungsaufgabe). Damit erfüllt H die Bedingungen in Definition 6.26, ist also eine Kontrollmatrix für $\mathscr{C}$.

Es bleibt zu zeigen, dass $m = N - K$ gilt, H also $N - K$ Zeilen hat. Nach Definition ist $\mathscr{C}^\perp = \mathrm{Im}(H)$ und $\mathscr{C} = \mathrm{Ker}(H)$; mt der Dimensionsformel folgt $N = \dim(\mathrm{GF}(2)^N) = \dim(\mathrm{Im}(H)) + \dim(\mathrm{Ker}(H)) = \dim(\mathscr{C}^\perp) + \dim(\mathscr{C}) = \dim(\mathscr{C}^\perp) + K$, also $\dim(\mathscr{C}^\perp) = N - K$ und damit $m = N - K$. □

Bemerkung 6.29 Auch wenn das Skalarprodukt $\langle \cdot, \cdot\rangle$ auf $\mathrm{GF}(2)^N$ äquivalent zum euklidischen in $\mathbb{R}^N$ definiert ist, hat es auf endlichen Körpern andere Eigenschaften. Insbesondere gibt es Vektoren $0 \neq x \in \mathrm{GF}(2)^N$, die auf sich selbst orthogonal stehen, also $\langle x, x\rangle = 0$ erfüllen (z. B. $x = (1, 1) \in \mathrm{GF}(2)^2$), was im $\mathbb{R}^N$ unmöglich ist. Damit ist auch $\mathscr{C}^\perp$ kein Komplement im eigentlichen Sinne, da weder $\mathscr{C} \cap \mathscr{C}^\perp = \{0\}$ noch $\mathscr{C} \cup \mathscr{C}^\perp = \mathrm{GF}(2)^N$ garantiert sind. Es gibt sogar sogenannte *selbstduale Codes* mit $\mathscr{C}^\perp = \mathscr{C}$, auf die wir hier jedoch nicht näher eingehen. ◁

Mit einer Kontrollmatrix können wir leicht überprüfen, ob ein N-Wort $y \in \mathrm{GF}(2)$ ein Codewort ist, indem wir die Bedingung $Hy^T = 0$ überprüfen. Außerdem können wir Aussagen über die Minimaldistanz von H ableiten. Die *Hamming-Codes*, die wir gleich definieren, werden über ihre Kontrollmatrix definiert. An deren Struktur lässt sich ablesen, dass Hamming-Codes Minimaldistanz 3 haben, woraus wir dann ihre Perfektheit folgern werden.

Definition 6.30 (Hamming-Codes) Zusätzlich zum bereits bekannten $(7,4)$-Hamming-Code definieren wir nun für jedes $m \geq 2$ einen Hamming-Code $\mathscr{C}$ mit $N = 2^m - 1$, $K = 2^m - m - 1$, und $\mathrm{d}(\mathscr{C}) = 3$ wie folgt: Sei H eine $m \times N$-Matrix, deren Spalten (in beliebiger Reihenfolge) alle Elemente von $\mathrm{GF}(2)^m \setminus \{0\}$ enthalten. Ein Code, dessen Kontrollmatrix H ist, nennen wir *Hamming-Code.* ◁

Lemma 6.31 *Sei $\mathscr{C}$ ein Hamming-Code zum Parameter $m \geq 2$.*

1. $\mathrm{d}(\mathscr{C}) = 3$.
2. *$\mathscr{C}$ ist perfekt.* ◁

Beweis. 1. Seien $h_1, \ldots, h_N$ die Spalten von H. Für $x \in \mathscr{C}$ gilt dann

$$Hx^T = \sum_{i\colon x_i \neq 0} h_i = 0$$

nach Definition 6.26. Wir zeigen $\mathrm{d}(\mathscr{C}) = \mathrm{w}(\mathscr{C}) = 3$ indem wir erst $\mathrm{w}(\mathscr{C}) \in \{1,2\}$ ausschließen und dann ein $x \in \mathscr{C}$ mit $\mathrm{w}(x) = 3$ konstruieren.

Wäre $\mathrm{w}(\mathscr{C}) = 1$, gäbe es ein $x \in \mathscr{C}$ mit $x_k = 0$ für ein $k \in \{1, \ldots, N\}$ und $x_i = 0$ sonst. Dann wäre $0 = Hx^T = h_k$, es gäbe also eine Nullspalte in H, was nach Definition von H nicht stimmt. Wäre $\mathrm{w}(\mathscr{C}) = 2$, gäbe es $x \in \mathscr{C}$ und $k \neq l$ mit

$$x_i = \begin{cases} 1 & i = k \text{ oder } i = l,, \\ 0 & \text{sonst.} \end{cases}$$

und damit $0 = Hx^T = h_k + h_l$, also $h_k = -h_l = h_l$ (da $1 = -1$), es gäbe also zwei gleiche Spalten in H, was ebenso nicht stimmt.

Seien nun i_1, i_2, i_3 die Spaltenindizes mit

$$h_{i_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, h_{i_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, h_{i_3} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

und $\hat{x} \in \mathrm{GF}(2)^N$ definiert durch

$$\hat{x}_i = \begin{cases} 1 & i \in \{i_1, i_2, i_3\} \\ 0 & \text{sonst.} \end{cases}$$

so dass $Hx^T = h_{i_1} + h_{i_2} + h_{i_3} = 0$, also $\hat{x} \in \mathscr{C}$ gilt. Wegen $\mathrm{w}(\hat{x}) = 3$ gilt $\mathrm{w}(\mathscr{C}) \leq \mathrm{w}(\hat{x}) = 3$, mit $\mathrm{w}(\mathscr{C}) > 2$ von oben also $\mathrm{w}(\mathscr{C}) = \mathrm{d}(\mathscr{C}) = 3$.

2. Wir rechnen die Bedingung von Satz 6.14 nach:

$$\begin{aligned} 2^K \sum_{i=0}^{1} \binom{N}{i} &= 2^K \left(\binom{N}{0} + \binom{N}{1} \right) \\ &= 2^{2^m - m - 1}(1 + 2^m - 1) \\ &= 2^{2^m - m - 1} \cdot 2^m = 2^{2^m - 1} = 2^N. \end{aligned}$$

□

6.3. Reed-Solomon-Codes

Reed-Solomon-Codes finden Anwendung in digitaler Fernseh- und Radioübertragung (DVB, DAB), auf Audio CDs oder QR-Codes. Auch wenn sie heutzutage langsam durch LDPC- und Turbo-Codes abgelöst werden, da für diese in den letzten Jahren besonders effiziente Decodieralgorithmen entwickelt wurden, gehören sie nach wie vor zu den wichtigsten und verbreitetsten Codes überhaupt.

Bemerkung 6.32 Für $n \in \mathbb{N}$ sei $\mathrm{GF}(2^n)$ der endliche Körper mit 2^n Elementen (siehe Satz 2.19) und $N \in \mathbb{N}$. Dann ist $\mathrm{GF}(2^n)^N$ wie bisher ein Vektorraum, nur dass die Einträge eines Vektors jetzt nicht mehr Bits aus $\mathrm{GF}(2)$, sondern Körperelemente von $\mathrm{GF}(2^n)$ sind.

1. Ein linearer (N, K)-Blockcode über $\mathrm{GF}(2^n)$ ist ein linearer Unterraum von $\mathrm{GF}(2^n)^N$ der Dimension K.
2. Der Hamming-Abstand von $x, y \in \mathrm{GF}(2^n)^N$ ist wie bisher definiert als

$$\mathrm{d}(x, y) = |\{i \in \{1, \dots, N\} : x_i \neq y_i\}|$$

   nur dass jetzt $x_i, y_i \in \mathrm{GF}(2^n)$ sind. Genauso direkt verallgemeinern sich $\mathrm{d}(\mathscr{C})$, $\mathrm{w}(\mathscr{C})$ etc.

Ist $\mathscr{C}$ ein linearer (N, K)-Blockcode über $\mathrm{GF}(2^n)$, erhalten wir einen (nicht notwendigerweise linearen) binären $(n \cdot N, n \cdot K)$-Blockcode $\tilde{\mathscr{C}}$, indem wir mit Notation 2.22 jedes Element von $\mathrm{GF}(2^n)$ als n-Vektor über $\mathrm{GF}(2)$, also n Bits, darstellen und diese aneinanderhängen. Wir können also auch mit nicht-binären Codes ohne große Umstände die Daten einer binären Quelle codieren und über einen binären Kanal verschicken.

Ist zum Beispiel $\mathscr{C}$ ein linearer $(3, 2)$-Code über $\mathrm{GF}(2^4)$ und $x = (\alpha_1, \alpha_2, \alpha_3) \in \mathscr{C}$, wobei $\alpha_i \in \mathrm{GF}(2^4)$ die Vektor-Darstellungen

$$\alpha_1 = (1, 1, 0, 1), \alpha_2 = (0, 0, 1, 0), \alpha_3 = (0, 1, 1, 0)$$

haben, so ist das entsprechende Codewort $\tilde{x}$ des entsprechenden $(4 \cdot 3, 4 \cdot 2)$-Codes $\tilde{\mathscr{C}}$ über $\mathrm{GF}(2)$ gegeben durch $\tilde{x} = 110100100110$.

Warum tun wir uns den Umweg über $\mathrm{GF}(2^n)$ dann überhaupt an? Der Grund ist, dass man im nicht-binären Fall leichter „gute“ Codes mit starken Eigenschaften (z. B. hohe Minimaldistanz) findet – beispielsweise die gleich definierten Reed-Solomon-Codes. ◁

Vektor	Potenz von α
00	–
01	α^0
10	α^1
11	α^2

$\cdot$	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

Tabelle 6.1.: Der Körper $\mathrm{GF}(2^2)$. Links: Elemente in Vektordarstellung und als Potenz Erzeugers. Rechts: Multiplikationstabelle

Wir benötigen noch ein Resultat aus der Algebra, das wir ohne Beweis zitieren:

Satz 6.33 *Für jedes $n \in \mathbb{N}$ besitzt $\mathrm{GF}(2^n)^\times = \mathrm{GF}(2^n) \setminus \{0\}$ einen* Erzeuger*, also ein Element $\alpha \in \mathrm{GF}(2^n)$ mit*

$$\left\{\alpha^0 (= 1), \alpha^1, \alpha^2, \dots, \alpha^{2^n-2}\right\} = \mathrm{GF}(2^n) \setminus \{0\},$$

wobei $\alpha^k = \underbrace{\alpha \cdots \alpha}_{k\text{-mal}}$. Außerdem gilt $\beta^{2^n-1} = 1$ für alle $\beta \in \mathrm{GF}(2^n)^\times$. ◁

Beispiel 6.34 Mit $n = 2$ erhalten wir den Körper $\mathrm{GF}(2^2)$ mit 4 Elementen, die wir als Vektoren $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$ schreiben können, und nach Notation 2.22 entspricht die Addition in $\mathrm{GF}(2^2)$ der Addition auf $\mathbb{Z}_2^2$, z. B. $(0,1) + (1,1) = (1,0)$. Man kann zeigen, dass $\alpha = (1,0)$ ein Erzeuger von $\mathrm{GF}(2^2)$ mit $\alpha^0 = (0,1)$ und $\alpha^2 = (1,1)$ ist. Dadurch ist dann auch die Multiplikation auf $\mathrm{GF}(2^2)$ bestimmt: sind $x = \alpha^i, y = \alpha^j \in \mathrm{GF}(2^2)^\times$, gilt $x \cdot y = \alpha^{i+j} = \alpha^{i+j \pmod{2^n-1}}$, z. B. $(1,0) \cdot (1,1) = \alpha^1 \cdot \alpha^2 = \alpha^3 = \alpha^0 = 1$. Für $x = (0,0)$ oder $y = (0,0)$ ist auch $x \cdot y = 0$, so dass wir alle Produkte haben (siehe Tabelle 6.1). ◁

Definition 6.35 (Polynome) Seien $n, K \in \mathbb{N}$ mit $K \leq 2^n$. Sei $\mathrm{GF}(2^n)[t]$ die Menge aller Polynome

$$f = a_r t^r + a_{r-1} t^{r-1} + \cdots + a_1 t + a_0$$

in der Variablen t mit Koeffizienten $a_i \in \mathrm{GF}(2^n)$ (siehe Definition 4.25). Man kann zeigen, dass $\mathrm{GF}(2^n)[t]$ ein Vektorraum über dem Körper $\mathrm{GF}(2^n)$ ist. Sei

$$\begin{aligned}\mathrm{GF}(2^n)[t]_{K-1} &:= \{f \in \mathrm{GF}(2^n)[t] \colon \deg f \leq K-1\} \\ &= \left\{a_{K-1} t^{K-1} + a_{K-2} t^{K-2} + \cdots + a_1 t + a_0 \colon a_i \in \mathrm{GF}(2^n), i = 0, \dots, K-1\right\}\end{aligned}$$

die Einschränkung auf Polynome vom Grad $\deg f \leq K-1$. ◁

Lemma 6.36 *$\mathrm{GF}(2^n)[t]_{K-1}$ ist ein K-dimensionaler Unterraum von $\mathrm{GF}(2^n)[t]$.* ◁

Definition 6.37 (Reed-Solomon-Code) Seien $n, N, K \in \mathbb{N}$ mit $K \leq N \leq 2^n$ gegeben. Sei α ein Erzeuger von $\mathrm{GF}(2^n)^\times$. Die Menge

$$\mathscr{C} := \left\{f(1), f(\alpha), \dots, f\left(\alpha^{N-1}\right) : f \in \mathrm{GF}(2^n)[t]_{K-1}\right\} \subseteq \mathrm{GF}(2^n)^N$$

nennen wir den *Reed-Solomon-Code* $\mathrm{RS}(n, N, K)$. Die Elemente von $\mathscr{C}$ entstehen also dadurch, dass wir irgendein Polynom $f \in \mathrm{GF}(2^n)[t]_{K-1}$ an den N Stellen $1, \alpha, \dots, \alpha^{N-1}$ auswerten und die Ergebnisse zu einem N-Wort über $\mathrm{GF}(2^n)$ aneinanderhängen. ◁

Bemerkung 6.38 Aus der Definition von RS(n, N, K) erhalten wir unmittelbar eine Codierungsfunktion $E_{\mathscr{C}}\colon \mathrm{GF}(2^n)^K \to \mathrm{GF}(2^n)^N$: jedes $w = (w_1, \dots, w_K) \in \mathrm{GF}(2^n)^K$, können wir eindeutig mit einem Polynom $w(t) \coloneqq \sum_{i=0}^{K-1} w_{i+1} t^i \in \mathrm{GF}(2^n)[t]_{K-1}$ identifizieren (die K Einträge von w sind also die Koeffizienten des Polynoms) und dann $E_{\mathscr{C}}(w) = \left(w(1), w(\alpha), \dots, w\left(\alpha^{N-1}\right)\right)$ setzen. ◁

Beispiel 6.39 Sei $n = 2$, $N = 3$ und $K = 2$. Dann ist $\mathrm{GF}(2^n)[t]_{K-1} = \mathrm{GF}(2^2)[t]_1$ die Menge aller Polynome vom Grad höchstens 1:

$$\mathrm{GF}(2^2)[t]_1 = \left\{a_1 t + a_0 \colon a_1, a_0 \in \mathrm{GF}(2^2)\right\}.$$

Sei z. B. $w = ((1,0),(1,1)) = (\alpha, \alpha^2) \in \mathrm{GF}(2^2)^K$. Dann ist $w(t) = \alpha + \alpha^2 \cdot t \in \mathrm{GF}(2^2)[t]_1$ das entsprechende Polynom, und

$$\begin{aligned} E_{\mathscr{C}}(w) &= \left(w(1), w(\alpha), w(\alpha^2)\right) \\ &= \left(\alpha + \alpha^2 \cdot 1, \alpha + \alpha^2 \cdot \alpha, \alpha + \alpha^2 \cdot \alpha^2\right) \\ &= ((1,0) + (1,1), (1,0) + (0,1), (1,0) + (1,0)) \\ &= ((0,1),(1,1),(0,0)) = x \in \mathrm{GF}(2^2)^N \end{aligned}$$

das Codewort zu w mit der Binärdarstellung $x = 011100$. ◁

Lemma 6.40 *Seien $1 \le K \le N \le 2^n$ gegeben und $\mathscr{C}$ =RS(n, N, K) der entsprechende Reed-Solomon-Code. Dann ist $\mathscr{C}$ ein linearer (N, K)-Blockcode über $\mathrm{GF}(2^n)$ mit Minimaldistanz $\mathrm{d}(\mathscr{C}) = N - K + 1$ und somit ein MDS-Code.* ◁

Beweis. Sei

$$\begin{aligned} \varphi\colon \mathrm{GF}(2^n)[t]_{K-1} &\to \mathrm{GF}(2^n)^N \\ f &\mapsto \left(f(1), f(\alpha), \dots, f\left(\alpha^{N-1}\right)\right) \end{aligned}$$

die „Auswertungsabbildung“, die ein Polynom an den Elementen $1, \alpha, \dots, \alpha^{N-1}$ auswertet. Diese Abbildung ist linear, da für $f_1, f_2 \in \mathrm{GF}(2^n)[t]_{K-1}$

$$\begin{aligned} \varphi(f_1 + f_2) &= \left((f_1 + f_2)(1), \dots, (f_1 + f_2)\left(\alpha^{N-1}\right)\right) \\ &= \left(f_1(1) + f_2(1), \dots, f_1\left(\alpha^{N-1}\right) + f_2\left(\alpha^{N-1}\right)\right) \\ &= \varphi(f_1) + \varphi(f_2) \end{aligned}$$

und analog $\varphi(\lambda f_1) = \lambda \varphi(f_1)$ für $\lambda \in \mathrm{GF}(2^n)$ gilt. Da $\mathscr{C} = \varphi(\mathrm{GF}(2^n)[t]_{K-1})$ das Bild von $\mathrm{GF}(2^n)[t]_{K-1}$ unter φ ist und $\mathrm{GF}(2^n)[t]_{K-1}$ nach Lemma 6.36 ein Unterraum ist, muss auch $\mathscr{C}$ ein Unterraum, also ein linearer Code sein. Wenn wir zeigen können dass φ injektiv ist, haben wir dann nach der Dimensionsformel (Lineare Algebra) auch $\dim(\mathscr{C}) = \dim\left(\varphi(\mathrm{GF}(2^n)[t]_{K-1})\right) = \dim(\mathrm{GF}(2^n)[t]_{K-1}) = K$ nach Lemma 6.36 gezeigt. Seien also $f, f' \in \mathrm{GF}(2^n)[t]_{K-1}$ mit

$$\begin{aligned} \varphi(f) = \varphi(f') &\Leftrightarrow \varphi(f) - \varphi(f') = 0 \\ &\Leftrightarrow \left(f(1) - f'(1), \dots, f\left(\alpha^{N-1}\right) - f'\left(\alpha^{N-1}\right)\right) = 0 \\ &\Leftrightarrow \left((f - f')(1), \dots, (f - f')\left(\alpha^{N-1}\right)\right) = 0 \\ &\Leftrightarrow (f - f')(1) = \cdots = (f - f')\left(\alpha^{N-1}\right) = 0, \end{aligned}$$

das Polynom $f - f'$ hat also N verschiedene Nullstellen. Da aber $\deg(f - f') \leq K - 1$ und $K - 1 < N$ nach Voraussetzung, impliziert dies nach Korollar 2.42 (ein von 0 verschiedenes Polynom f hat höchstens $\deg(f)$ Nullstellen) dass $f - f' = 0$, also $f = f'$ und damit φ injektiv, also auch $\dim(\mathscr{C}) = K$ ist.

Es bleibt die Aussage über $\mathrm{d}(\mathscr{C})$ zu zeigen. Da (wie wir eben schon verwendet haben) für jedes $0 \neq f \in \mathrm{GF}(2^n)[t]_{K-1}$ höchstens $K-1$ Einträge von $\varphi(f)$ Null sein können, müssen mindestens $N - (K-1)$ Einträge ungleich 0 sein, woraus $\mathrm{w}(\mathscr{C}) \geq \mathrm{w}(\varphi(f)) \geq N - K + 1$ folgt. Definieren wir andererseits

$$\hat{f}(t) = \prod_{i=0}^{K-2} (t - \alpha^i) \in \mathrm{GF}(2^n)[t]_{K-1},$$

hat $\hat{f}$ die $K-1$ Nullstellen $1, \alpha, \ldots, \alpha^{N-1}$, so dass nach demselben Argument wie oben $\hat{f}$ an den $N - K + 1$ Stellen $\alpha^{K-1}, \ldots, \alpha^{N-1}$ nicht Null sein kann, also $\mathrm{w}(\varphi(\hat{f})) = N - K + 1$ und damit insgesamt $\mathrm{w}(\mathscr{C}) = N - K + 1$ gilt. Da $\mathscr{C}$ die Singleton-Schranke (Satz 6.17) mit Gleichheit erfüllt, ist er ein MDS-Code. □

6.4. Fallbeispiel: QR-Codes

Ein *QR-Code* ist ein zweidimensionaler Barcode zur Repräsentation digitaler Daten, die in einer quadratischen Pixelmatrix als Schwarz-Weiß-Muster abgebildet werden, das zur automatischen Erfassung beispielsweise per Smartphone-Kamera geeignet ist, aber auch in der Produktionslogistik weite Verbreitung findet. Der ISO-Standard 18004 [ISO06] spezifiziert für verschiedene Datenmengen 40 Versionen in aufsteigender Größe (21 × 21 Pixel (Version 1) bis 177 × 177 Pixel (Version 40)). Für jede Version stehen vier verschiedene Fehlerkorrektur-Levels L, M, Q, H zur Verfügung, die mittels Reed-Solomon-Codes die Korrektur von etwa 7% (L) bis 30% (H) Lesefehlern (beispielsweise bei Verschmutzung des Symbols oder schlechten Lichtverhältnissen beim Abfotografieren) erlauben.

Es stehen vier Datenmodi zur Verfügung, welche die effiziente Binärcodierung verschiedener Datentypen festlegen:

- *Numeric:* Dezimalzahlen, also Ziffernfolgen
- *Alphanumeric:* Lateinische Großbuchstaben, Ziffern, einige Sonderzeichen
- *Bytes:* beliebige Binärdaten, beispielsweise UTF-8-codierter Text mit deutschen Sonderzeichen; vgl. Abschnitt 2.2
- *Kanji:* japanische/chinesische Zeichen

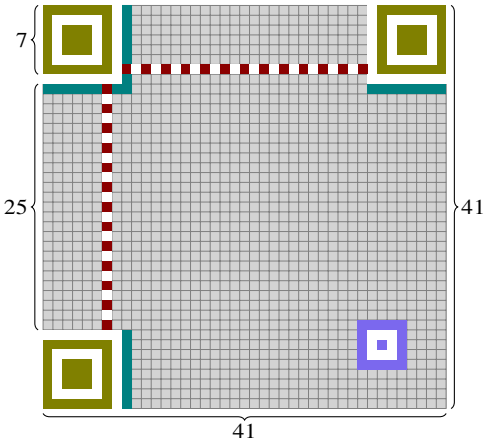


Abbildung 6.3.: Aufbau eines QR-Codes Version 6M. Positionsmuster (incl. Trennstreifen) und Ausrichtungsmuster sowie Synchronisationsmuster sind fest. Die Format-Information speichert Fehlerlevel (hier: M) und Maske in codierter Form. Der übrige Platz wird für die RS-codierten Nutzdaten verwendet.

Die maximale Kapazität im jeweiligen Modus ist durch Version (Größe) und Fehler-Level bestimmt. Wir behandeln exemplarisch Version 6M (41 × 41 Pixel, ≈15% Fehlerkorrektur) im *Alphanumeric*-Modus, die bis zu 154 Zeichen aufnehmen kann, was für URLs, Kontaktdaten oder kurze Texte (ohne Sonderzeichen) ausreicht.

6.4.1. Aufbau des QR-Codes

Wir stellen den QR-Code als Matrix $Q \in \mathbb{F}_2^{41\times 41}$ dar, wobei eine 1 für ein schwarzes, eine 0 für ein weißes Pixel steht. Die Indizierung beginnt nach ISO-Standard bei 0, so dass $Q_{0,0}$ die linke obere und $Q_{40,40}$ die rechte untere Ecke ist. Der vorhandene Platz von 41 × 41 Pixeln ist in verschiedene Bereiche unterteilt (Abbildung 6.3):

1. Immer gleiche *Positionierungs-Muster* enthalten keine Information, sondern helfen den Bilderkennungs-Algorithmen beim korrekten Auffinden des QR-Codes in einem Foto. Es gibt:

   - Drei 7 × 7-Positions-Muster aus konzentrischen Quadraten mit den Mittelpunkten $Q_{3,3}$, $Q_{37,3}$, $Q_{3,37}$ erlauben das Finden des QR-Codes im Foto; da es in der rechten unteren Ecke fehlt, kann Rotation erkannt werden. Nach innen sind die Muster jeweils durch 1 Pixel breite weiße Trenner abgegrenzt.

   - Ein 5 × 5-Ausrichtungs-Muster mit Mittelpunkt bei $Q_{34,34}$ (größere QR-Code-Versionen enthalten mehr davon).

Zeichen:	0	...	9	A	B	...	Z	*Leerz.*	$	%	*	+	-	.	/	:
Wert:	0	...	9	10	11	...	35	36	37	38	39	40	41	42	43	44

Tabelle 6.2.: Zuordnung der erlaubten alphanumerischen Zeichen zu einem Zahlwert zwischen 0 und 44.

- Synchronisations-Muster: je eine Zeile bzw. Spalte abwechselnd 1/0-Pixel in Spalte und Zeile 6 zwischen den Positions-Mustern. Erleichtert die Synchronisation zwischen Bild-Koordinaten und Matrix-Indizes bei Verzerrung (z. B. schräge Aufnahme).

Zusätzlich sollte der gesamte QR-Code mit einer mindestens 4 Pixel breiten weißen Randzone umgeben sein.

2. Format-Information: Meta-Information über verwendete Fehlerkorrektur und *Maske* (siehe später), inkl. Fehlerkorrektur.

3. Die eigentlichen Nutzdaten, die zunächst binär codiert und dann mit einem Reed-Solomon-Code über $\mathrm{GF}(2^8)$ gegen Fehler abgesichert werden.

Von den 41^2 Pixeln sind $3 \cdot 8^2 + 5^2 + 2 \cdot 25 + 31$ „besetzt“, für die codierten Nutzdaten bleiben somit 1383 bit übrig.

6.4.2. Quellencodierung: von Zeichen zu Bits

Im alphanumerischen Modus stehen 45 verschiedene Zeichen zur Verfügung, die wir nach Tabelle 6.2 mit dem Alphabet $A = \{0, \dots, 44\}$ identifizieren. Die zu speichernde Information können wir daher als Wort

$$w = w_1 w_2 \cdots w_k \in A^k$$

mit $k := l(w) \leq 154$ Zeichen darstellen. Sei $\mathrm{bin}_s(x) : \{0, \dots, 2^s - 1\} \to \mathrm{GF}(2)^s$ die Darstellung einer natürlichen Zahl als s Bit lange Binärzahl (z. B. $\mathrm{bin}_5(6) = 00101$). Die fw_i werden nun zu $\lfloor k/2 \rfloor$ Paaren aus jeweils zwei Zahlen zusammengefasst:

$$w = ((w_1, w_2), (w_3, w_4), \dots),$$

wobei für l ungerade am Ende noch w_k einzeln übrig bleibt. Diese Paare werden nun mit der injektiven Abbildung

$$\begin{aligned} E_Q : A \times A &\to \mathrm{GF}(2)^{11} \\ (z_1, z_2) &\mapsto \mathrm{bin}_{11}(45 \cdot z_1 + z_2) \end{aligned}$$

binär codiert und die Ergebnisse aneinandergehängt. Ist k ungerade, wird das letzte Zeichen w_k zu einer 6-Bit-Zahl $\mathrm{bin}_6(w_k)$.

Bemerkung 6.41 Ganz im Sinne von Satz 5.28 codieren wir paarweise, um weniger Kapazität zu verschwenden: zur einzelnen Codierung eines $a \in A$ bräuchten wir 6 Bits, da $|A| = 45 > 2^5 = 32$, mit 5 Bits geht es also nicht. Andererseits können wir Paare $(a, b) \in A^2$ mit 11 Bits codieren, da $\left|A^2\right| = 45^2 = 2025 < 2^{11} = 2048$. Der Unterschied 2048/2025 ist so klein, dass es sich nicht lohnt, noch mehr Zeichen zusammenzufassen. ◁

Der so berechneten Binärfolge wird noch das feste 4-Tupel 0010, das den alphanumerischen Modus ankündigt, sowie der *Zeichenzähler* $\mathrm{bin}_9(k)$ (damit der Empfänger später weiß, wie lang die eigentliche Nachricht ist) vorangestellt und die *Endmarke* 0000 angehängt; insgesamt erhalten wir die binäre Nachricht

$$m(w) = \underbrace{0010}_{\text{Modus}}\ \underbrace{\mathrm{bin}_9(k)}_{\text{Zeichenzähler}}\ \underbrace{E_Q(w_1, w_2)\ E_Q(w_3, w_4) \cdots [\mathrm{bin}_6(w_k)]}_{\text{Daten}}\ \underbrace{0000}_{\text{Endmarke}},$$

wobei $\mathrm{bin}_6(w_k)$ nur für ungerade k vorkommt. Da $k \leq 154$ ist die Bitlänge maximal $l(m(w)) \leq 4 + 9 + (154/2) \cdot 11 + 4 = 864$.

Beispiel 6.42 Die Nachricht `AC-42` soll codiert werden. Laut Tabelle 6.2 entspricht dies der Zahlenfolge $w = (10, 12, 41, 4, 2)$, $k = l(w) = 5$. Wir erhalten die Paare $(10, 12)$, $(41, 4)$ und den Rest (2), und somit $E_Q(10, 12) = \mathrm{bin}_{11}(45 \cdot 10 + 12) = \mathrm{bin}_{11}(462) = 00111001110$, $E_Q(41, 4) = 11100111001$ und $\mathrm{bin}_6(2) = 000010$. Da $k = 5$, ist der Zeichenzähler $\mathrm{bin}_9(5) = 000000101$ und die gesamte Nachricht somit

$$m(\texttt{AC-42}) = 0010000000101001110011101110011100100001000000.$$ ◁

6.4.3. Umrechnung in Bytes

Um einen RS-Code über $\mathrm{GF}(2^8)$ zu verwenden, wird die im letzten Abschnitt erzeugte binäre Nachricht $m(w)$ in *Bytes*, also Blöcke von jeweils 8 bit unterteilt. Falls $8 \nmid l(m(w))$, werden dazu weitere 0er an $m(w)$ angehängt, bis die Länge ein Vielfaches von 8 ist. Wir erhalten somit ein „Byte-Wort“ $b(m) = b_1, \ldots, b_n$, $b_i \in \mathrm{GF}(2)^8$. Ein QR-Code Version 6M kann insgesamt 108 Bytes an Nutzdaten aufnehmen. Ist $n < 108$, werden zum Auffüllen abwechselnd die festen Bytes 11101100 und 00010001 angehängt, so dass am Ende genau 108 Bytes $b = b_1, \ldots, b_{108}$ an Nutzdaten vorliegen.

6.4.4. Reed-Solomon-Codierung

Satz 6.43 *Sei $1 \leq K \leq N \leq 2^n$, α ein Erzeuger von $\mathrm{GF}(2^n)$ und $g(t) := \prod_{i=0}^{N-K}(t - \alpha^i)$. Für $w = (w_1, \ldots, w_K) \in \mathrm{GF}(2^n)^K$ sei wie in Bemerkung 6.38 $w(t) = \sum_{i=0}^{K-1} w_{i+1} t^i$ und $r^w(t)$ der Rest bei Polynomdivision von $t^{N-K} \cdot w(t)$ durch $g(t)$, also $t^{N-K} \cdot w(t) = g(t) \cdot q(t) + r^w(t)$ für ein Polynom $q(t)$ und $\deg(r^w(t)) < \deg(g(t)) = N - K$. Dann ist*

$$\begin{aligned} E_{\mathscr{C}} \colon \mathrm{GF}(2^n)^K &\to \mathrm{GF}(2^n)^N \\ w &\mapsto \left(w_1, \ldots, w_k, r_1^w, \ldots, r_{N-K}^w\right) \end{aligned}$$

eine Codierungsfunktion für RS(n, N, K). ◁

Im Gegensatz zu Bemerkung 6.38 müssen wir also nur *eine* Polynomdivision ausführen und dann die Koeffizienten des Restpolynoms an die Eingabe w anhängen, um das Wort w mit einem Reed-Solomon-Code zu codieren. Eine Codierungsfunktion $E_{\mathscr{C}}$ der Form $E_{\mathscr{C}}(w) = (w, f(w))$, bei der jedes Codewort in den ersten K Komponenten gerade der Eingabe entspricht, nennt man *systematisch*.

Die 108 Byte $b = b_1 \cdots b_{108}$ aus dem vorigen Abschnitt werden jetzt in vier Gruppen $\tilde{b}_1, \ldots, \tilde{b}_4$ zu je 27 Byte zusammengefasst ($27 \cdot 4 = 108$). Nach Notation 2.22 fassen wir jedes $\tilde{b}_i$ als Element von $\mathrm{GF}(2^8)^{27}$ auf, das nach der Methode aus Satz 6.43 mit dem RS(8, 43, 27)-Code zu einem Codewort $c_i = (\tilde{b}^i, \tilde{r}^i) \in \mathrm{GF}(2^n)^43$ mit $\tilde{r}^i \in \mathrm{GF}(2^n)^{16}$ codiert wird. Nach Lemma 6.40 hat dieser Code die Minimaldistanz $43 - 27 + 1 = 17$ und kann so auf dem BSC bis zu $(d-1)/2 = 8$ Fehler oder $8/43 \approx 18{,}6\,\%$ korrigieren. Die vier Codewörter $c^1, \ldots, c^4$ können damit insgesamt bis zu 32 Fehler korrigieren – aber nur, wenn sich diese „gleichmäßig“ verteilen, so dass in jedem Codewort genau 8 Fehler passieren. Landen hingegen alle Fehler im selben Codewort, reichen schon 9 Fehler (und damit $9/143 \approx 6{,}3\,\%$) für einen Decodierfehler.

6.4.5. Verteilung der Bytes in Q

Um zu vermeiden, dass alle Fehler im selben Codewort vorkommen, werden die Daten der vier Codewörter $c^1, \ldots, c^4$ vor der Platzierung in der Matrix Q „durcheinandergewürfelt“. Die Idee ist, dass Fehler beispielsweise durch Flecken oder Verdeckung eines Teils des QR-Codes oft auf einem kleinen Bereich der Matrix gehäuft auftreten. Das Durcheinanderwürfeln sorgt dann dafür, dass die Codewörter trotzdem etwa gleichmäßig von diesen Fehlern betroffen sind. Sei $\tilde{b}^i = (b^i_1, \ldots, b^i_{27})$ und $r^i = (r^i_1, \ldots, r^i_{16})$. Die endgültige Byte-Sequenz $x \in \mathrm{GF}(2^8)^{172}$ erhalten wir nach der Vorschrift

$$x = b^1_1 \cdots b^4_1 \cdots b^1_{27} \cdots b^4_{27} r^1_1 \cdots r^4_1 \cdots r^1_{16} \cdots r^4_{16};$$

die Einträge der folgenden Tabelle werden also spaltenweise von oben nach unten und von links nach rechts aneinandergehängt.

$\tilde{b}^i$				r^i			
b^1_1	b^1_2	…	b^1_{27}	r^1_1	r^1_2	…	r^1_{16}
b^2_1	b^2_2	…	b^2_{27}	r^2_1	r^2_2	…	r^2_{16}
b^3_1	b^3_2	…	b^3_{27}	r^3_1	r^3_2	…	r^3_{16}
b^4_1	b^4_2	…	b^4_{27}	r^4_1	r^4_2	…	r^4_{16}

Nun ist jedes x_i, $1 \le i \le 172$, ein Element von $\mathrm{GF}(2^8)$, das wir wieder nach Notation 2.22 als Byte $x_{i,1}, \ldots, x_{i,8}$, also 8 bit aus $\mathrm{GF}(2)^8$ schreiben. Die 172 Bytes werden nun zweispaltig schlangenförmig von unten rechts nach oben links in der Matrix Q verteilt, wobei Einträge, die durch Positionsmuster etc. schon belegt sind, einfach übersprungen werden. Ein $x_i \in \mathrm{GF}(2^8)$ belegt so in der Regel eine 2×4-Teilmatrix von Q (siehe Abbildung 6.4).

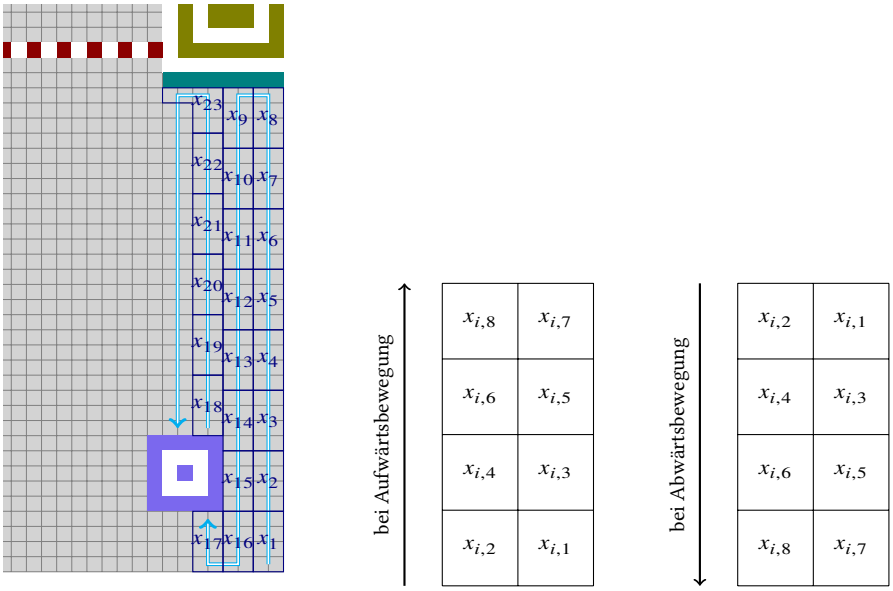


Abbildung 6.4.: Verteilung der finalen Datensequenz $x \in \mathrm{GF}(2^8)^{172}$ in Q. Links: $x_1, x_2, \ldots$ werden schlangenförmig von unten beginnend in den noch nicht durch Positionsmuster o. ä. belegten Platz verteilt, wobei jeweils zwei Spalten gemeinsam verwendet werden. Mitte und rechts: Verteilung der 8 Bits des i-ten Bytes x_i bei Auf- und Abwärtsbewegung der eben beschriebenen Schlangenlinien.

Bei 172 Bytes werden so insgesamt $172 \cdot 8 = 1376$ Einträge von Q belegt, so dass von den 1383 Pixeln im Datenbereich am Ende 7 unbelegt (weiß) bleiben.

6.4.6. Maske und Format-Information

Nach Platzierung von x in der Matrix Q können durch Zufall Muster auftreten, die den Positions-Mustern ähneln und so die Bilderkennung erschweren. Um dies zu verhindern, wird eine von acht *Muster-Matrizen* $M^0, \ldots, M^7 \in \mathrm{GF}(2)^{41\times 41}$ ausgewählt und auf Q addiert. Dabei wird nur der Datenbereich von Q verändert: $M^k_{i,j} = 1$ kann nur gelten, vonn $Q_{i,j}$ nicht zu einem Auffindungs-Muster oder der Format-Information gehört. Beim Lesen des QR-Codes wird dieser Schritt wieder rückgängig gemacht, indem das entsprechende M^k nochmal auf die Summe $Q + M^k$ addiert wird, denn $Q + M^k + M^k = Q$.

Am Ende wird das verwendete Fehlerlevel (bei uns: M) und die Nummer der verwendeten Maske $k \in \{0, \ldots, 7\}$ zu einem binären 5-Tupel f zusammengefasst, z. B. $f = \underbrace{00}_{\mathrm{M}}\ \underbrace{101}_{\mathrm{bin}_3\, k}$ und mit einem $(15, 5)$-Code $\hat{\mathscr{C}}$ mit Minimaldistanz $\mathrm{d}(\hat{\mathscr{C}}) = 7$ zu einer 15-Bit-Nachricht $\varphi = E_{\hat{\mathscr{C}}}(f)$ codiert. Darauf wird schließlich noch die Folge $\gamma = 101010000010010$ addiert; dadurch erreicht man, dass die finale Format-Information $\vartheta = \varphi + \gamma$ nie der Nullvektor ist (in diesem Fall wäre er schwer vom Trenner um die Positionsmuster zu unterscheiden). Schließlich wird ϑ *zweimal* in die

Bereiche um die Positionsmuster geschrieben. Diese Information, ohne die keine Decodierung des QR-Codes möglich ist, ist also besonders gut gesichert.

6.4.7. Decodierung

Zur Decodierung werden alle genannten Schritte in umgekehrter Reihenfolge durchgeführt: Nachdem ein Bilderkennungs-Algorithmus den QR-Code aufgefunden und gedreht bzw. entzerrt hat, wird zunächst die Format- und Maskeninformation ausgelesen, die Maske rückgängig gemacht, die vier Codewörter extrahiert, jedes davon mit einem Reed-Solomon-Decodieralgorithmus (den wir hier nicht behandelt haben) decodiert, woraus sich die Nachricht $m(w)$ ergibt. Anhand von Modus-Indikator 0010 und Zeichenzähler kann schließlich die Originalnachricht rekonstruiert werden, solange in jedem der vier Codewörter höchstens 8 Lesefehler vorliegen.

Literatur

[Ash90] Robert B. Ash. *Information Theory*. Dover, 1990.

[BH12] Mohamed Barakat und Timo Hanke. *Cryptography — Lecture notes*. 2012. URL: http://www.mathematik.uni-kl.de/~barakat/Lehre/WS10/Cryptography/lecture_notes/Cryptography.pdf.

[Buc99] Johannes Buchmann. *Einführung in die Kryptographie*. Springer, 1999.

[Gat10] Andreas Gathmann. *Algebraische Strukturen*. 2010. URL: http://www.mathematik.uni-kl.de/agag/mitglieder/professoren/gathmann/notes/agstr/.

[HPS08] Jeffrey Hoffstein, Jill Pipher und Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer, 2008.

[ISO06] ISO/IEC. *Information technology – Automatic identification and data capture techniques – QR Code 2005 bar code symbology specification*. ISO Standard 18004. ISO/IEC, Sep. 2006.

[KK10] Christian Karpfinger und Hubert Kiechle. *Kryptologie*. Vieweg+Teubner, 2010.

[KW08] Götz Kersting und Anton Wakolbinger. *Elementare Stochastik*. Birkhäuser Verlag AG, 2008. URL: http://link.springer.com/book/10.1007/978-3-7643-8431-9.

[Mac03] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. URL: http://www.inference.phy.cam.ac.uk/itprnn/book.html.

[Mar10] Thomas Markwig. *Elementare Zahlentheorie*. 2010. URL: http://www.mathematik.uni-kl.de/~keilen/download/LectureNotes/zahlentheorie.pdf.

[SS08] Harald Scheid und Wolfgang Schwarz. *Elemente der Arithmetik und Algebra*. Spektrum, 2008.

[Wil08] Wolfgang Willems. *Codierungstheorie und Kryptographie*. Birkhäuser, 2008. URL: http://link.springer.com/book/10.1007/978-3-7643-8612-2.

Stichwortverzeichnis