Kryptologie

Mannheim, 5. April 2017

Prof. Dr. A. Wiedemann
Berufsakademie Mannheim
Coblitzweg 7
68163 Mannheim
wiedemann@ba-mannheim.de
http://cruncher.ba-mannheim.de/~wiedeman

Inhaltsverzeichnis

I Kryptologie **1**

1 Grundbegriffe und Überblick **3**
1.1 Kryptologie – Abgrenzung und Ziele 4
1.1.1 Grundlegendes 4
1.1.2 Das grundlegende Szenario 8
1.1.3 Sicherheit von Kryptosystemen 13
1.1.4 Angriffsarten 15
1.1.5 Das Kerckhoffs Prinzip 18
1.1.6 Superencryption 19
1.2 Verschlüsselungsarten 20
1.2.1 Symmetrische Verschlüsselung 20
1.2.2 Asymmetrische Verschlüsselung 23
1.2.3 Hybridverfahren 27
1.3 Who is who in der Kryptographie 29
1.4 Software 33
1.5 Review Questions 34

2 Klassische Kryptosysteme **35**
2.1 Skytale 36
2.2 Shift Chiffre 37
2.3 Die Affine Chiffre 40
2.4 Die Vigenère Chiffre 47
2.5 Die Playfair Chiffre 51
2.6 Die Hill Chiffre 56
2.7 Die ADFGVX Chiffre 60
2.8 Permutations Chiffre 65
2.9 Produkt Chiffren 68
2.10 One–Time Pad 71
2.11 Rotor Maschinen 83
2.11.1 Aufbau und Arbeitsweise der Enigma 84
2.11.2 Schlüsselraum der Enigma 87
2.11.3 Einsatz der Enigma 89
2.11.4 Entschlüsselung der Enigma-Chiffre 90
2.11.5 Blechtley Park 94
2.12 Übungen 95

3 Symmetrische Kryptosysteme **97**
3.1 Generelle Eigenschaften von symmetrischen Chiffren 97

3.1.1 Block Chiffren und Strom Chiffren . . . 97
3.1.2 Diffusion und Konfusion . . . 99
3.1.3 Beispiele symmetrischer Chiffren . . . 100
3.2 Die Feistel Chiffre . . . 103
3.2.1 Verschlüsselung im Feistel Netzwerk . . . 104
3.2.2 Feistel Entschlüsselungsalgorithmus . . . 107
3.3 Der Data Encryption Standard . . . 111
3.3.1 Ein Demo–Modell . . . 115
3.3.2 Reales DES . . . 125
3.4 Betriebsmodi von Blockchiffren . . . 134
3.4.1 ECB–Mode . . . 135
3.4.2 CBC–Modus . . . 140
3.4.3 CFB–Mode . . . 142
3.4.4 OFB–Modus . . . 144
3.4.5 Der Counter Modus . . . 145
3.5 Unix Passwörter . . . 146

4 Weitere symmetrische Kryptosysteme 149
4.1 International Data Encryption Algorithm . . . 149
4.1.1 Designprinzipien . . . 149
4.1.2 IDEA Verschlüsselung . . . 153
4.1.3 IDEA Dechiffrierung . . . 157
4.2 Blowfish . . . 162
4.2.1 Erzeugung der Rundenschlüssel und S-Boxen . . . 162
4.2.2 Ver- und Entschlüsselung . . . 165
4.2.3 Zusammenfassung . . . 167
4.3 RC4 und WEP . . . 169
4.3.1 Die RC4 Stromchiffre . . . 169
4.3.2 WEP . . . 175
4.3.3 Funktionsweise von WEP . . . 175
4.4 Der RC5 Algorithmus . . . 178
4.4.1 RC5 Parameter . . . 179
4.4.2 RC5 Schlüsselerzeugung . . . 179
4.4.3 The RC5 Encryption Algorithm . . . 181
4.4.4 RC5 Entschlüsselung . . . 182
4.4.5 Modes . . . 183

5 Der Advanced Encryption Standard – AES 185
5.1 Vereinfachter Rijndael–Algorithmus . . . 188
5.1.1 Überblick . . . 188
5.1.2 S–AES Ver– und Entschlüsselung . . . 192
5.2 Das Rijndael Kryptosystem . . . 197
5.2.1 Die ByteSub Transformation . . . 201
5.2.2 Die ShiftRow Transformation . . . 202
5.2.3 Die MixColumn Transformation . . . 202
5.2.4 The Round Key Addition . . . 202
5.3 Key Schedule . . . 202

6 Public Key Kryptosysteme 205
6.1 Einführung . . . 205

6.2 Public–Key Kryptosysteme 208
6.2.1 Allgemeine Eigenschaften von Public–Key Verfahren . . . 208
6.2.2 Anwendungen von Public–Key Kryptosystemen 214
6.2.3 Anforderungen an Public–Key Kryptosysteme 215
6.2.4 Beispiel einer Trap-door Funktion 218
6.3 Der RSA Algorithmus 223
6.4 Aspekte des RSA-Algorithmus 230
6.4.1 Ver– und Entschlüsselung 230
6.4.2 Schlüsselerzeugung 231
6.4.3 Sicherheitsaspekte des RSA Algorithmus 232
6.5 Das Diffie-Hellman Key Exchange Verfahren 234
6.6 ElGamal Kryptosysteme 240
6.6.1 Digitale Signatur mit dem ElGamal Verfahren 241
6.6.2 Verschlüsselung mit dem ElGamal Kryptosystem 244

7 Zero-Knowledge Protokolle 255
7.1 Die grundlegende Idee 255
7.2 Das Fiat-Shamir Protokoll 258
7.3 Zero-Knowledge Beweis für die Kenntnis eines diskreten Logarithmus 262

8 Digital Cash 263
8.1 Elektronisches Bargeld 265
8.2 Weitere grundlegende Protokolle 272
8.2.1 Blinde Signaturen 272
8.2.2 Secret Splitting 276
8.2.3 Bit-Commitment Protokolle 277
8.3 Das Protokoll von D. Chaum, A. Fiat und M. Naor 279
8.4 Das Protokoll von Stefan Brands 281

9 Weitere Protokolle 285
9.1 The Dining Cryptographers Problem 285
9.2 Oblivious Transfer 286

10 Authentifikation und Hash Funktionen 287
10.1 Ziele der Authentifikation 290
10.2 Möglichkeiten zur Authentifikation 292
10.2.1 Nachrichtenverschlüsselung 292
10.2.2 Message Authentication Code 297
10.2.3 Hash Funktionen 300
10.3 Weiterführendes über Hash Funktionen 305
10.3.1 Generelle Anforderungen an Hash Funktionen 305
10.3.2 Einfache Hash Funktionen 306
10.3.3 Geburtstagsangriff 310
10.4 Der MD5 Hash Algorithmus 316
10.5 Der Secure Hash Algorithm SHA 326
10.5.1 Die Logik des SHA-1 Algorithmus 326
10.5.2 Die SHA-1 Kompressions Funktion 329
10.5.3 Vergleich von SHA-1 und MD5 331
10.5.4 Angriff auf SHA-1 331

11 Elliptische Kurven Kryptographie **333**
11.1 Einführung 333
11.2 Elliptische Kurven über den reellen Zahlen 335
11.2.1 Sukzessives Verdoppeln 355
11.3 Elliptische Kurven über $\mathbb{Z}_p$ 357
11.4 Faktorisierung mit elliptischen Kurven 378
11.5 Kryptosysteme auf elliptischen Kurven 383
11.5.1 Codierung von Klartext 383
11.5.2 Elliptische Kurven ElGamal Kryptosysteme 385
11.5.3 Elliptische Kurven Diffie-Hellman Key Exchange 394
11.5.4 ElGamal digitale Signatur 395

12 Quantenkryptographie **397**
12.1 Ein Experiment mit Quanten 399
12.2 Schlüsselaustausch mit Quantenkryptographie 404

II Mathematische Grundlagen **407**

13 Algebraische Strukturen **409**
13.1 Halbgruppen 410
13.2 Gruppen 412
13.2.1 Beispiele 413
13.3 Ringe 421
13.4 Körper 423

14 Elementare Eigenschaften von Zahlen **425**
14.1 Teiler 425
14.1.1 Übungen 427
14.2 Primzahlen 428
14.3 Der größte gemeinsame Teiler zweier Zahlen 431
14.4 Relativ prime Zahlen 433
14.5 Die Eulersche Totientenfunktion 434

15 Kongruenzen **439**
15.1 Kongruenzrelationen 439
15.2 Restklassen 442
15.3 Elementare Eigenschaften von Restklassen 450
15.3.1 Modulare Arithmetik 450
15.3.2 Division 456
15.3.3 Modulare Exponentiation 458
15.4 Der Chinesische Restesatz 467
15.5 Endliche Körper 472

16 Gruppen **475**
16.1 Überblick 475
16.2 Gruppenhomorphismen 484
16.3 Zyklische Gruppen 490
16.4 Untergruppen 494

17 Sätze von Fermat und Euler **501**
17.1 Der kleine Fermatsche Satz 501
17.2 Der Satz von Euler 504
17.3 Quadratische Reste 508
17.4 Quadratwurzeln modulo n 514

18 Bestimmung des ggT zweier Zahlen **523**
18.1 Der Euklidische Algorithmus 523
18.2 Der erweiterte Euklidische Algorithmus 534
18.2.1 Der erweiterte Euklidische Algorithmus 535
18.3 Der Algorithmus von Josef Stein 546
18.3.1 Übungen 550

19 Primzahlen **551**
19.1 Faktorisierung 551
19.1.1 Die RSA Challenge 553
19.1.2 Die Teilermethode 555
19.1.3 Pollards Rho Algorithmus 563
19.2 Pseudozufallszahlen 572
19.2.1 Zyklische Verschlüsselung 575
19.2.2 ANSI X9.17 Psudozufallszahlengenerator 576
19.2.3 Blum-Blum-Shub Pseudozufallszahlengenerator 578
19.3 Weiteres über Primzahlen 583
19.3.1 Umrechung großer Zahlen 583
19.3.2 Der Primzahlsatz 585
19.3.3 Primzahltests 587

20 Diskreter Logarithmus **601**
20.1 Primitive Wurzeln 601
20.2 Berechnung diskreter Logarithmen 615
20.2.1 'Brute-Force' Verfahren 616
20.2.2 Der Algorithmus von Shanks 616
20.2.3 Der Pohlig-Hellman Algorithmus 619

21 Komplexität von Berechnungen **621**
21.1 Die Länge einer Zahl 621
21.2 Laufzeitbestimmungen 623
21.2.1 Bit-Operationen 624
21.2.2 Algorithmen 626
21.3 Laufzeitbestimmung des Euklidischen Algorithmus 632
21.3.1 Von polynomialer zur exponentieller Laufzeit 637
21.4 $\mathcal{P}$, $\mathcal{NP}$ und $\mathcal{NP}$ vollständige Probleme 640
21.4.1 Instanzen von Problemen, Suchprobleme und Entscheidungsprobleme 640
21.4.2 $\mathcal{P}$ und $\mathcal{NP}$ 644
21.4.3 Reduktion eines Problems auf ein anderes 646
21.4.4 $\mathcal{NP}$-vollständige Probleme 648

22 Galois Felder **651**
22.1 Endliche Körper der Form $\mathtt{GF}(p)$ 653

22.1.1 Endliche Körper der Ordnung p 653
22.1.2 Berechnen des multiplikativen Inversen in $\mathtt{GF}(p)$ 654
22.2 Arithmetik mit Polynomen 656
22.2.1 Gewöhnliche Arithmetik mit Polynomen 656
22.2.2 Polynom–Arithmetik mit Koeffizienten in $\mathbb{Z}_p$ 659
22.2.3 Euklidischer Algorithmus für Polynome 665
22.2.4 Erweiterter Euklidischer Algorithmus 669
22.2.5 Irreduzible Polynome 677
22.3 Endliche Körper der Form $\mathtt{GF}(2^n)$ 688
22.3.1 Motivation 688
22.3.2 Modulare Polynom Arithmetik 691
22.4 Der endliche Körper $GF(2^4)$ 693
22.4.1 Polynome über $GF(2^4)$ 697
22.5 Linear Feedback Shift Register 716

23 Linear rückgekoppelte Sequenzen 725
23.1 Feedback Shift Register, Periodizität 725
23.2 Impuls Response Sequenzen, Charakteristische Polynome 744

A Lösungen zu den Übungen aus Kapitel [14.1.1] 753
A.1 Übung [14.1] 753

B Lösungen zu den Übungen aus Kapitel [18.3.1] 755
B.1 Übung [18.2] 755
B.2 Übung [18.3] 756
B.3 Übung [18.4] 757
B.4 Übung [18.5] 758

C Akronyme 759

D Glossar 763

Bibliography 769

Teil I

Kryptologie

Kapitel 1

Grundbegriffe und Überblick

Das Ziel dieses einführenden Kapitels ist es, einen Überblick der Verfahren zur Verschlüsselung kennenzulernen und einen Eindruck über die Stärken heutiger Verschlüsselungsverfahren zu gewinnen. Es gibt eine Vielzahl sehr guter Bücher zu diesem Thema, nichttechnische Aspekte der Kryptologie findet man beispielsweise in [123, 191, 204] oder [136]. Die ultimative Quelle der historischen Betrachtung der Kryptologie ist DAVID KAHNs monumentales Werk *The Codebreakers* [118]. Technische und mathematische Darstellungen der Kryptologie findet man in den Büchern [33, 55, 78, 154, 205, 213, 215, 218] und in BRUCE SCHNEIERs epochalem Werk *Applied Cryptography* [193].[1] Die rigorose mathematische Diskussion der Kryptologie findet man in den beiden Büchern von NEAL KOBLITZ [129, 130], empfehlenswert ist auch der Übersichtsartikel von RON RIVEST [181].

Grundsätzlich sind die Methoden, Daten zu verschlüsseln nichts anderes als Methoden, Daten geeignet zu codieren, also beispielsweise ähnlich zu den Verfahren, die angewendet werden, um Fehler während einer Datenübertragung zu erkennen und/oder zu korrigieren. Das Ziel von fehlererkennenden und –korrigierenden Codierungen ist also, auf einem verrauschten Übertragungskanal verlorengegangene Informationen wiederzugewinnen. Datenverschlüsselung hat ein etwas anderes Ziel: Die Daten müssen in einer systematischen Weise so verändert werden, dass nicht autorisierte Personen auf die darin enthaltenen Informationen nicht zugreifen können.

Aufgrund dessen benutzen Verschlüsselungsverfahren völlig andere Methoden als die fehlerkorrigierenden bzw. fehlererkennenden Verfahren. Die zugrunde liegenden mathematischen Methoden sind jedoch sehr ähnlich. Moderne Verschlüsselungsverfahren benutzen unter anderem Eigenschaften von Primzahlen, endlichen Körpern und elliptischen Kurven.

[1] Zitat: ... *The book the National Security Agency wanted never be published.*

1.1 Kryptologie – Abgrenzung und Ziele

1.1.1 Grundlegendes

Wir betrachten zunächst die banal scheinende Frage:

Was ist Kryptologie?

Auf diese Frage gibt es zwei Antworten: Eine kurze und eine lange Antwort.

Die kurze Antwort:

Kryptologie ist die Wissenschaft der Verschlüsselung von Daten.

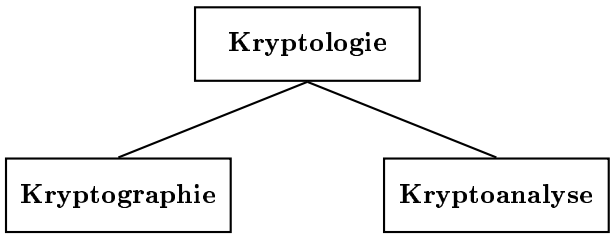
Die lange Antwort:

> Kryptologie ist eine Wissenschaft, die sich mit Methoden der Verschlüsselung und damit verwandte Verfahren befasst. Das Ziel dabei ist, Daten gegenüber unautorisierter Manipulation zu schützen.

Die Personen, die die Kryptologie praktizieren sind die **Kryptologen**, heutzutage sind dies high–level Mathematiker.

Kryptologie kann auch charakterisiert werden als das Studium sicherer Kommunikation. Die Kryptologie umfasst dabei die beiden Teildisziplinen:

- ☞ **Kryptographie**
- ☞ **Kryptoanalyse**



Die **Kryptographie**[2] bezeichnet denjenigen Zweig der Kryptologie, der sich mit der Erstellung und dem Design von Algorithmen (Verfahren) zur Ver– und Entschlüsselung von Daten beschäftigt. Das dabei verfolgte Ziel ist unter anderem, die *Geheimhaltung* von Nachrichten und deren *Authentizität* zu gewährleisten. Die andere Richtung, *i.e.* das nicht autorisierte Brechen eines Verschlüsselungsverfahrens — oder die Entwicklung genereller Methoden, wie Verschlüsselungsverfahren geknackt werden können — nennt man **Kryptoanalyse**.

[2] Das Wort Kryptographie läßt sich zurückführen auf das griechische Wort `kryptos`, was verstecken bedeutet, und `graphikos`, was schreiben bedeutet [112, p. 2].

Der Kryptologie liegt ein einfaches **Kommunikationsmodell** zugrunde. In diesem Modell gibt es zwei Protagonisten — traditionell heissen diese **Alice** und **Bob** — die über einen unsicheren Kommunikationskanal kommunizieren möchten. Diese Kommunikation soll vertraulich sein, *i.e.* eine von Alice gesendete Nachricht soll nur von Bob gelesen werden können (und umgekehrt). Dieser unsichere Kanal kann

- ☞ eine Telefonverbindung über Festnetz
- ☞ eine mobile Telefonübertragung
- ☞ eine Internetverbindung
- ☞ aber auch ein Speichervorgang auf einem Datenträger mit (späterem) Lesezugriff

sein. Natürlich ist auch ein Bösewicht im Spiel, traditionell ist dies in der Kryptologie eine Frau mit dem Namen **Eve**.[3] Die Absicht von Eve ist, die Kommunikation zwischen Alice und Bob abzuhören.

Dieses Kommunikationsmodell bildet beispielsweise folgende Szenarien ab:

1. Alice und Bob können zwei Personen sein, die über ein zelluläres Telefonnetz kommunizieren. Eve versucht, diese Kommunikation abzuhören.
2. Alice benutzt einen Web–Brouwser um ein Produkt eines Online–Anbieters (Bob) zu kaufen, der durch die Web–Site repräsentiert wird. Die Angreiferin Eve versucht den Traffic zwischen Web–Brouwser und Server abzuhören um so zu Kreditkarten Informationen von Alice zu gelangen. Oder Eve versucht sich für einen der beiden Kommunikationspartner auszugeben.
3. Alice sendet eine E–Mail über das Internet an Bob. Eine Angreiferin Eve versucht, die Nachricht zu lesen und einen Teil der Nachricht zu modifizieren. Oder Eve gibt sich als Alice aus und sendet ihre Nachrichten — als Alice ausgegeben — an Bob. Dieses Angreifer–Szenario nennt man **Man–in–the–middle Attacke**.
4. Betrachte das Szenario, bei dem Alice eine Smart Card an einem Terminal einer Bank benutzt, um Geld abzuheben. Die Smart Card dient unter anderem dazu, die Inhaberin (Alice) bei einem Mainframe Computer (Bob) einer Bank zu authentifizieren. In diesem Szenario versucht Eve, die Kommunikation zwischen Terminal und Mainframe abzuhören, um an Kontoinformationen von Alice zu gelangen. Oder Eve versucht sich als Alice auszugeben, um Geldbeträge von Alice Konto abzuheben.

Aus diesen Beispielen soll deutlich werden, dass die Kommunikationspartner — deren Kommunikation eine Absicherung gegenüber Eve Angriffen erfordert — nicht notwendigerweise Menschen sind. Dies können auch Computer, Smart Cards oder Software Module sein.

Präziser — wie in [193], [64] oder [107, p. 3] beschrieben — beschäftigt sich die Kryptographie mit der Untersuchung mathematischer Systeme um mehrere

[3] Dies stammt ab aus dem Englischen Begriff *eavesdropping* für lauschen.

sicherheitsrelevante Probleme zu lösen. Eine detaillierte Beschreibung der Ziele und Werkzeuge der Kryptologie findet man auch in dem Übersichtsartikel von RIVEST [181].

- ✘ **Geheimhaltung** (privacy)
  Ein kryptographisches System, das die Geheimhaltung einer Nachricht gewährleistet, verhindert die Entnahme von Informationen durch eine nicht autorisierte Person während der Übertragung über einen unsicheren Kommunikationskanal. Mit anderen Worten, der *Sender* einer Nachricht kann sicher sein, dass ausschließlich der legitime Empfänger die Nachricht lesen kann.

- ✘ **Authentizität**
  Ein kryptographisches System, das Authentizität gewährleistet, verhindert das nicht autorisierte Einschleusen einer Nachricht in eine unsicheren Kommunikationskanal. Einem Eindringling soll es nicht möglich sein, eine falsche Nachricht als richtige Nachricht auszugeben. Mit anderen Worten, der *Empfänger* einer Nachricht kann *sich selbst* davon überzeugen, dass die Nachricht vom angeblichen Sender stammt.

- ✘ **Signaturen**
  Der Empfänger einer Nachricht kann *eine neutrale dritte Partei davon überzeugen*, dass die erhaltene Nachricht von dem angeblichen Sender stammt.

- ✘ **Integrität**
  Ein kryptologisches System, das Integrität gewährleistet, verhindert die nicht autorisierte Manipulation des Inhalts einer Nachricht während der Übertragung über einen öffentlichen, unsicheren Kommunikationskanal. Sender und Empfänger einer Nachricht können sicher sein, dass die Nachricht während der Übertragung unverändert bleibt.

- ✘ **Verbindlichkeit** (non–repudiation)
  Ein Sender soll nicht im Nachhinein behaupten können, eine Nachricht nicht gesendet zu haben.

Anmerkungen:

① Die Eigenschaft der Authentizität bildet die Grundlage der **digitalen Signatur**. Signatur ist stärker als Authentifizierung.

② Die hier formulierten Ziele der Kryptographie sind essentielle Voraussetzungen für die Abbildung von Geschäftsprozessen im Internet.

Im Kontext der Kryptographie sind drei Begriffe klar zu differenzieren: *Identifikation, Authentifikation* und *Autorisierung* (siehe [92]).

- ✘ **Identifikation** bedeutet die Zuordnung einer Identität zu einem Subjekt.
- ✘ **Authentifizierung** bedeutet die Verifikation der Gültigkeit einer Tatsache, beispielsweise einer Identität.
- ✘ **Autorisierung** bedeutet die Zuordnung von Rechten zu einer Sache.

Alle drei Konzepte sind wichtig, können aber unabhängig voneinander zugeteilt werden. Es gibt jede Menge Situationen, die nicht gleichzeitig alle Anforderungen benötigen. Falls beispielsweise jemand mit einem 20 Euro-Schein einen Laib Brot in einer Bäckerei bezahlt, muß sich der Kunde zu diesem Zweck weder identifizieren noch authentifizieren. Falls der Warenkorb des Kunden außerdem eine Flasche Sekt beinhaltet, reichen die grauen Haare und Falten im Gesicht aus, zu beweisen, dass er 18 Jahre alt ist und damit legal Alkohol erwerben kann. Damit ist also die Authentifikation des Alters durchgeführt, die Notwendigkeit der Identifikation besteht nicht.

Betrachtet man folgendes Beispiel: Man fährt mit 65 km/h durch eine 30er Zone und wird dabei von der Polizei erwischt. Es besteht kein dringender Grund, die Identität des Polizisten festzustellen oder zu kennen, der die Anzeige aufnimmt. Er ist autorisiert die Anzeige aufzunehmen und die Authentifizierung als Polizist geschieht über die Uniform, Anzeigenformular und die Schusswaffe, die er trägt.

In diesem Zusammenhang ist es sinnvoll, die beiden Begriffe **Datenschutz** und **Datensicherheit** gegeneinander abzugrenzen.[4]

- **Datenschutz**[5]
  Bei dem Datenschutz geht es um die Privatsphäre eines jeden Menschen. Datenschutz garantiert jedem Bürger ein Recht auf informelle Selbstbestimmung und schützt ihn dadurch vor missbräuchlicher Verwendung seiner Daten. Für die Verarbeitung personenbezogener Daten gibt es Regeln, die in dem Bundesdatenschutzgesetz BDSG hinterlegt sind. Der Datenschutz befasst sich also mit der Frage, ob personenbezogene Daten überhaupt verarbeitet respektive gespeichert werden dürfen.

- **Datensicherheit**[6]
  Im Gegensatz zum Datenschutz befasst sich due Datensicherheit mit dem Schutz von Daten, dies geschieht unabhängig davon, ob diese personenbezogen sind oder nicht. Unter Datensicherheit fallen grundsätzlich auch Daten, die keinen Personenbezug haben. Datensicherheit bezieht sich sowohl auf analoge als auch digitale Daten.

[4] Siehe beispielweise C. Eckert, [74] Seite 5.
[5] Engl.: *privacy*.
[6] Engl.: *protection*.

Das Ziel von Datensicherheit ist, Sicherheitsrisikien zu begegnen und die Daten vor Manipulation, Verlust oder unberechtigter Kenntnisnahme zu schützen. Bei Datensicherheit geht es also nicht um die Frage, ob Daten überhaupt erhoben und verarbeitet werden dürfen (das ist die Frage des Datenschutzes), sondern um die Frage, welche Maßnahmen zum Schutz der Daten erhoben werden müssen.

Im folgenden klären wir einige grundlegende Begriffe der Kryptographie und sehen uns einige Definitionen an.

1.1.2 Das grundlegende Szenario

Angenommen, Alice möchte eine Nachricht über einen öffentlichen, unsicheren Kommunikationskanal an Bob senden. Diese Nachricht soll zudem vertraulich an Bob gesendet werden. Der Sender möchte verhindern, dass ein Lauscher (Eve) die Nachricht abhören kann.

Definition [1.1]:

Der **Klartext** (engl.: *plaintext, cleartext*) ist die Nachricht, die durch ein geeignetes mathematisches Verfahren in eine Form transformiert wird, in der eine Informationsentnahme durch eine nicht autorisierte dritte Partei verhindert wird.

Unter Klartext versteht man:

- Natürliche Sprachen wie Englisch, Deutsch etc.
- Im Zeitalter der Funkübertragung und Telegraphie: Durch MORSE Zeichen codierten Text einer natürlichen Sprache.
- Im Computerzeitalter: Durch Binärcode `0` und `1` geeignet codierter Text einer natürlichen Sprache.

Anmerkung:

Es gibt natürliche Sprachen, die ohne Verschlüsselung durchzuführen, einen hohen Grad an Sicherheit gewährleisten. Diese Erkenntnis nutzten die Amerikaner im Zweiten Weltkrieg an der Pazifikfront [151], indem Navajo–Indianer an beide Enden einer Kommunikationsleitung als Funker eingesetzt wurden. Die natürliche Sprache der Navajo Indianer ist mit keiner anderen Sprache verwandt, den Japanern war es bis Ende des Krieges nicht gelungen, den Navajo–Code zu brechen.

Es gibt zwei grundlegende Möglichkeiten, zu verhindern, dass eine nicht autorisierte Person Informationen aus einer Nachricht gewinnen kann.

- **Steganographie**
  Das Charakteristische der Steganographie (soviel wie *verdeckt schreiben*)

besteht darin (siehe [13, pp. 9–24]), die eigentliche Existenz einer Nachricht — nicht deren Inhalt — zu verschleiern bzw. zu verstecken. Bekannte steganographische Methoden sind die Verwendung von Geheimtinte oder die *Mikrophotographie*, mit der es möglich ist, eine DIN A4 Seite auf wenige Quadratmillimeter zu verkleinern. Steganographische Verfahren wurden bereits in der Antike verwendet, so findet man in KAHN ([118, pp. 81-82]) die Methode des HISTIAEUS, der einen seiner Sklaven kahlschor, die Nachricht auf dessen Kopfhaut schrieb und dann warten mußte, bis das Haar nachgewachsen war. Es ist auch möglich, eine geheime Nachricht in einer unverfänglichen offenen Nachricht zu verstecken, indem beispielsweise jeder zweite Buchstaben des unverfänglichen Texts die versteckte Nachricht ergibt.

Moderne Verfahren des digitalen Zeitalters benutzen digitale Bilder (Bitmaps) um dort Textinformationen zu verstecken.[7]

In einer Reihe von Anwendungsbereichen ist die Tatsache, dass kommuniziert wird nicht selbst schützenswert, hingegen jedoch etwa der Schutz vor Entfernung oder aber auch der Nachweis der Authentizität. Verfahren, die dies gewährleisten, werden in Analogie zu den in Papier eingebrachten Markierungen **digitale Wasserzeichen** genannt (siehe [69] oder [256]).

Anwendungsbereiche digitaler Wasserzeichen sind beispielsweise Annotationswasserzeichen, die Zusatzinformationen in Mediendaten einbetten, welche selbst bei Umwandlung in analoge Repräsentationen und weiterer Nachbearbeitung erhalten bleiben müssen, oder die Überwachung von Rundfunk– und Fernsehsendungen um etwa automatisch an Verwertungsgesellschaften abzuführende Gebühren zu erfassen oder umgekehrt die Ausstrahlung von Werbesendungen automatisch erfassen und nachweisen zu können.

Darüber hinaus sind digitale Wasserzeichen auch im Bereich des Urheberrechtsschutzes als Abschreckungsmerkmal oder in Verbindung mit kryptographischen Mechanismen wie elektronischen Signaturen und Zeitstempeldiensten zum Nachweis von Verletzungen urheberrechtlicher oder sonstiger vertraglicher Verpflichtungen einsetzbar. Zudem können digitale Wasserzeichen im Bereich der Authentifizierung von digitalen Daten und gedruckten Dokument eingesetzt werden, da robuste Wasserzeichen auch den Medienübergang zum gedruckten Dokument überstehen.

Sichtbare digitale Wasserzeichen sind nur von geringem Interesse, da sie leicht zu entfernen sind oder aber störend wirken. Primär von Interesse sind dahingegen Verfahren, welche nicht wahrnehmbar Datensätze in Mediendaten einbetten können, sodass diese robust gegenüber einer Reihe von bewussten und unbeabsichtigten Manipulationen wie etwa verlustbehafteter Kompression, Zurechtschneiden oder Verzerrungen sind. Darüber hinaus bieten fragile Wasserzeichen die Möglichkeit zur Manipulationserkennung, um die Integrität digitaler Medien nachzuweisen. Steganographische

[7] Siehe auch [74, pp. 283–287].

Verfahren, robuste als auch fragile Wasserzeichen werden meist mit kryptographischen Mechanismen gekoppelt, um die Sicherheit zu erhöhen.[8]

☞ **Kryptographie**
Im Gegensatz zur Steganographie verbergen die Methoden der Kryptographie nicht die Existenz der Nachricht sondern deren Inhalt. Durch geeignete Transformationen des Klartexts wird der eigentliche Inhalt einer Nachricht für Nichteingeweihte unlesbar.

Das Verfahren, eine Nachricht derart umzuwandeln, dass deren Inhalt versteckt wird, nennt man **Verschlüsselung**[9] oder **Chiffrierung**. Eine verschlüsselte Nachricht nennt man **Chiffretext**.[10]

Umgangssprachlich wird Verschlüsselung oftmals mit dem Begriff *Codierung* gleichgesetzt. Genauer betrachtet ist dies nicht korrekt, da die Codierung einer Nachricht einen anderen Zweck verfolgt als die Verschlüsselung. Daten und Informationen werden zu dem Zweck codiert, um sie geeignet verarbeiten zu können. Man denke an den ASCII Code. Dies stellt eine Codierung des Alphabets (und anderer Zeichen) dar in Form von 8 Bit Codeworten, mit dem Zweck, Texte in Rechnern be- und verarbeiten zu können. Primärer Zweck der Codierung ist es also nicht, den Inhalt einer Nachricht zu verbergen.

Den umgekehrten Vorgang, die Auflösung eines Chiffretexts in den Klartext, nennt man **Entschlüsselung** oder **Dechiffrierung**.

Ein **kryptographischer Algorithmus** — diesen nennt man auch **Chiffre**[11] — ist das mathematische Verfahren, das für die Ver- bzw. Entschlüsselung des Klartexts bzw. Chiffretexts eingesetzt wird.

Moderne kryptographische Verfahren verwenden einen **Schlüssel** (Engl.: *key*), üblicherweise mit dem Buchstaben K bezeichnet. Ein Schlüssel ist in der Regel ein von vielen möglichen (Zahlen-) Werten. Den Bereich der möglichen Werte eines Schlüssels nennt man den **Schlüsselraum**.[12]

In der Abbildung [1.1] ist das einfache Kommunikationsmodell skizziert, das der (symmetrischen) Kryptographie zugrunde liegt. Der Ablauf dieser Kommunikation erfolgt nach den folgenden Schritten:

1. Ein Klartext wird auf der Seite des Senders — hier Alice — erzeugt.
2. Der Klartext läuft durch den Verschlüsselungsalgorithmus und wird dadurch in den Chiffretext transformiert.
3. Essentiell dabei ist der geheime Schlüssel K, der in den Verschlüsselungsalgorithmus eingeht. Die Sicherheit dieses Kommunikationsmodells hängt

[8] Die Standard-Referenz für Digitale Wasserzeichen ist das Buch von Jana Dittman [68]. Siehe auch die Monographie von Klaus Schmeh [192].
[9] Engl.: *encryption*
[10] Engl.: *ciphertext*; synonyme Begriffe zu Chiffretext sind **Cryptogramm** oder **Cryptotext**
[11] Engl.: *chipher*
[12] Engl.: *keyspace*

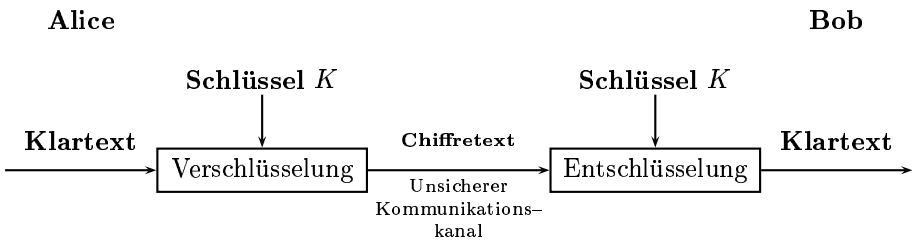


Abbildung 1.1: Symmetrische Verschlüsselung: Ver– und Entschlüsseln mit einem Schlüssel K.

dabei entscheidend davon ab, dass der Schlüssel K nur Alice und Bob bekannt ist.

4. Der Chiffretext wird über einen unsicheren Kommunikationskanal zum Empfänger übertragen.

5. Der Empfänger dechiffriert mit Hilfe eines geeigneten Dechiffrier Algorithmus und dem Schlüssel K den Chiffretext. Der Schlüssel K, den der Empfänger der Nachricht benutzt, ist der gleiche Schlüssel, den der Sender verwendet, um die Nachricht zu chiffrieren.

6. Schließlich liegt auf Empfängerseite der Klartext vor.

Ein auf diese Weise operierendes Kryptosystem nennt man **symmetrisches Verschlüsselungsverfahren**, da zum Ver- und Entschlüsseln der Nachricht bzw. des Chiffretexts der gleiche Schlüssel K benötigt wird.

Die meisten kryptographischen Systeme setzen zwei grundlegende Verfahren ein, um den Klartext in Chiffretext zu transformieren: **Substitution** und **Transposition**:

- **Substitution**
  Bei einer Substitutionsoperation wird das Klartextzeichen durch ein anderes Zeichen ersetzt. Die Position des Zeichens bleibt erhalten.
- **Transposition**
  Transpositionsoperationen — diese nennt man auch **Permutationen** — vertauschen die Reihenfolge der Zeichen, *i.e.* die Positionen der Klartextzeichen innerhalb des Klartextes. Die Form des Zeichens selbst bleibt dabei erhalten und wird nicht verändert.

Moderne Verschlüsselungsalgorithmen verwenden geeignete Kombinationen von Substitutions– und Permutationsoperationen.

Die entwickelten Konzepte formalisiert man um sie einer mathematischen Beschreibung zugänglich zu machen. Dies führt auf den Begriff des **Kryptosystems**, eingeführt von Claude Shannon [199].

Definition [1.2]:

Ein **Kryptosystem** $\mathcal{K}$ ist ein Fünf–Tupel

$$\mathcal{K} = (P, C, K, E, D)$$

mit:

① P ist eine endliche Menge, das Klartextalphabet.

② C ist eine endliche Menge, das Chiffretextalphabet.

③ K ist der Schlüsselraum, dies ist endliche Menge der möglichen Schlüssel.

④ Zu jedem Wert $k \in K$ gibt es einen *Verschlüsselungsalgorithmus* $e_k \in E$ und einen zugeordneten Entschlüsselungsalgorithmus $d_k \in D$. Jedes $e_k \in E$ mit:

$$e_k : P \longrightarrow C$$

und jedes $d_k \in D$ mit

$$d_k : C \longrightarrow P$$

erfüllen

$$d_k(e_k(x)) = x \quad \forall\, x \in P$$

Das Design eines Kryptosystems erfordert also gemäß dieser Definition von Claude Shannon die Angabe von fünf Parametern:

- ☞ Das *Klartextalphabet*, *i.e.* das Alphabet, über dem Klartexte gebildet werden.
- ☞ Das *Chiffretextalphabet*, *i.e.* das Alphabet (das können auch mehrere sein), über dem die Chiffretexte gebildet werden.
- ☞ Der *Schlüsselraum*, das ist die Menge der möglichen Schlüssel des Kryprosystems.
- ☞ Das *Verschlüsselungsverfahren*, *i.e.* der Algorithmus, der Klartexte in Chiffretexte umwandelt.
- ☞ Das *Entschlüsselungsverfahren*, *i.e.* der zum Verschlüsselungsverfahren inverse Algorithmus, der den Chiffretext wieder in den Klartext zurücktransformiert.

Im Kapitel [2] werden wir eine Reihe klassischer Kryptosysteme unter diesen Aspekten kennenlernen.

1.1.3 Sicherheit von Kryptosystemen

Das ultimative Ziel bei dem Design eines guten Kryptosystems besteht darin, dass die Ver– und Entschlüsselungsoperation effektiv und kostengünstig[13] von den Anwendern des Kryptosystems eingesetzt werden kann. Andererseits jedoch muß das Kryptosystem einem kryptoanalytischen Angriff widerstehen — zumindest muß es so gestaltet sein, dass ein erfolgreicher Angriff nur mit nicht vertretbarem, ökonomischem Aufwand gelingt.

Die Kryptosysteme werden hinsichtlich Sicherheit in verschiedene Klassen eingeteilt. Man unterscheidet dabei:

- ☞ **Computational Security**
  Ein Kryptosystem, das sicher ist aufgrund unvertretbar hoher Kosten einer erfolgreichen kryptoanalytischen Attacke, welches jedoch einen Angriff mit unbegrenzt zur Verfügung stehenden Rechenleistung nicht widersteht, nennt man *computationally secure*. Die meisten eingesetzten Kryptosysteme sind von dieser Art, da sie praktikabel implementierbar sind.

  Man definiert ein Kryptosystem als computationally secure, falls der beste Algorithmus zum Brechen des Kryptosystems mindestens N Rechenoperationen erfordert. N ist dabei irgendeine feste, sehr große Zahl. Das Problem hierbei ist, dass für kein bekanntes, in der Praxis eingesetzte Kryptosystem nachgewiesen werden kann, dass es unter dieser Definition sicher ist. In der Praxis wird immer die Sicherheit eines Kryptosystems in Bezug auf eine bestimmte Angriffsart — *e.g.* Brute–Force Attacke, *i.e.* das Ausprobieren aller möglichen Schlüssel — untersucht. Wichtig ist zu realisieren, dass die Sicherheit gegenüber einer Angriffsart noch keine Sicherheit gegenüber einer anderen Angriffsart bietet.

[13] Der Terminus *kostengünstig* bezieht sich in diesem Kontext auf zu leistenden Aufwand hinsichtlich Rechenzeit und Speicherplatz, um die Ver– und Entschlüsselungsoperationen auszuführen.

☞ **Beweisbare Sicherheit**
Bei einem anderen Zugang reduziert man die Sicherheit eines Kryptosystems auf eine Reihe wohlverstandener mathematischer Probleme, von denen man allgemein annimmt, dass sie schwierig sind. Beispielsweise kann eine Aussage des Typs *ein gegebenes Kryptosystem ist sicher, falls eine gegebene Zahl n in einem vernünftigen Zeitraum nicht faktorisiert werden kann* bewiesen werden. Kryptosysteme dieser Art nennt man mitunter *beweisbar sicher*. Dieser Zugang liefert jedoch nur einen Beweis an Sicherheit relativ zu einem anderen Problem, also keinen absoluten Beweis der Sicherheit.

☞ **Uneingeschränkte Sicherheit**
Ein Kryptosystem, das jeder denkbaren kryptoanalytischen Attacke widersteht — ungeachtet der Frage, wie viel Rechenleistung zur Verfügung steht — nennt man *uneingeschränkt sicher*. Diese Systeme wurden von Claude Shannon in [199] untersucht. Das einzige Kryptosystem, das uneingeschränkte Sicherheit bietet und gelegentlich eingesetzt wird, ist das **One–Time Pad**.

1.1.4 Angriffsarten

Kryptosysteme müssen eine Reihe von Angriffsarten widerstehen. Diese unterscheidet man grob in vier Stufen und gliedert diese dahingehend, über welche Informationen der Angreifer verfügt um einen Angriff auszuführen:

- ✗ Ciphertext–only Attacke
- ✗ Known–plaintext Attacke
- ✗ Chosen–plaintext Attacke
- ✗ Chosen–ciphertext Attacke

Das ultimative Ziel eines Kryptoanalytikers ist in allen Fällen, weitere Chiffretexte zu entschlüsseln ohne zusätzliche Informationen. Der Idealfall für den Kryptoanalytiker tritt dann ein, wenn es ihm gelingt, den geheimen Schlüssel zu extrahieren. Jede Angriffsart setzt realistischerweise voraus, dass der Angreifer komplette Kenntnis über den verwendeten Verschlüsselungsalgorithmus hat.

Gute Kryptosysteme müssen den folgenden Angriffsarten widerstehen:

- **Ciphertext–only Attacke**
  Eine Ciphertext–only Attacke ist ein kryptoanalytischer Angriff, bei dem der Kryptoanalyst den Chiffretext mehrerer Nachrichten kennt jedoch nicht den zugehörigen Klartext. Diese Nachrichten wurden mit dem gleichen, bekannten Verschlüsselungsalgorithmus chiffriert. Die Aufgabe des Kryptoanalysten besteht darin, so viele Nachrichten wie möglich zu entschlüsseln, optimal ist es, den oder die verwendeten Schlüssel abzuleiten, um zukünftige Nachrichten entschlüsseln zu können.

  Eine Ciphertext–only Attacke kann daher folgendermaßen charakterisiert werden:

  > Gegeben sind i Chiffretexte
  >
  > $$C_1 = E_K(M_1), C_2 = E_K(M_2), \ldots, C_i = E_K(M_i)$$
  >
  > Finde entweder die zugehörigen Klartexte $M_1, \ldots, M_i$, den Schlüssel K oder einen Algorithmus, der die $i+1$te Nachricht M_{i+1} aus $C_{i+1} = E_K(M_{i+1})$ ableitet.

  Diese Angriffsart ist die schwierigste für den Kryptoanalytiker, da er hierbei über die wenigsten Informationen verfügt. Dennoch ist es die am häufigsten praktizierte Angriffsart, da der Angreifer in Regel keinerlei Kenntnis über den Inhalt einer abgehörten, verschlüsselten Nachricht hat.

  Ein Beispiel einer Ciphertext–only Attacke ist die **Brute Force Attacke**. Der Angreifer verfügt über einen abgehörten, chiffrierten Text und ist in Kenntnis des verwendeten Verschlüsselungsalgorithmus (⟶ KERCKHOFFS Prinzip). Durch das systematische Ausprobieren aller möglichen Schlüssel wird versucht, den Klartext zu gewinnen, bzw. den passenden Key K.

- **Known–plaintext Attacke**
  Eine Known–plaintext Attacke ist ein kryptoanalytischer Angriff, bei dem der Kryptoanalytiker einen bedeutenden Anteil von Chiffretext und zugehörigem Klartext zur Verfügung steht. Die Aufgabe des Kryptoanalytikers ist auch hier, entweder den — oder die Schlüssel abzuleiten, oder ein Verfahren zu finden, wie mit dem gleichen Schlüssel chiffrierte Nachrichten entschlüsselt werden können.

  Gegeben sind i passende Klartext/Chiffretextpaare

  $$M_1, C_1 = E_K(M_1), M_2, C_2 = E_K(M_2), \ldots, M_i, C_i = E_K(M_i)$$

  Finde entweder den Schlüssel K oder einen Algorithmus, um M_{i+1} aus $C_{i+1} = E_K(M_{i+1})$ abzuleiten.

  Eine Known–plaintet Attacke ist eine vielversprechende Angriffsart, falls sich Nachrichten — oder Teile einer Nachricht — wiederholen. Falls Alice immer den gleichen Header benutzt, wenn sie eine verschlüsselte E–Mail an Bob sendet, oder beginnt ihre E–Mail immer mit der Phrase *lieber Bob* oder beendet die Mail mit *Mit freundlichen Grüßen* — und der Angreifer ist sich dessen bewußt — dann sind dem Angreifer zumindest Teile des Klartextes bekannt.

  Ein Beispiel eines Known–Plaintext Angriffs wurde im 2. Weltkrieg aus dem Wüstenkrieg in der Sahara berichtet. Ein isolierter deutscher Außenposten meldete befehlsgemäß jeden Tag mit exakt der gleichen Nachricht, dass keine besonderen Vorkommnisse vorliegen. Diese gleiche Nachricht wurde jeden Tag mit einem anderen Schlüssel chiffriert. Auf diese Art und Weise verfügten die Alliierten jeden Tag über ein passendes Klartext–Chiffretext–Paar, das sich als extrem hilfreich erwies, den Schlüssel K für die Heeres Enigma abzuleiten. General Montgomery auf alliierter Seite tat gut daran, diesen Außenposten in Ruhe zu lassen, um die Übertragungen nicht zu stören.

- **Chosen–plaintext Attacke**
  Eine Chosen–plaintext Attacke ist ein kryptoanalytischer Angriff, bei dem der Angreifer eine unbegrenzte Anzahl von Klartextnachrichten seiner eigenen Wahl verwendet, diese verschlüsselt und den resultierenden Chiffretext untersuchen kann. Der Angreifer hat demnach nicht nur Zugriff auf den Chiffretext und den zugehörigen Klartext, er kann sogar den Klartext wählen, der verschlüsselt wird. Dies ist eine weitaus effektivere Angriffsart als die Known–plaintext Attacke, da der Kryptoanalyst spezielle Klartextnachrichten seiner Wahl benutzen kann, um den Schlüssel K des Verfahrens zu finden.

  Eine Chosen–plaintext Attacke kann wie folgt beschrieben werden:

  Gegeben sind die chiffrierten Nachrichten

  $$C_1 = E_K(M_1), C_2 = E_K(M_2), \ldots, C_i = E_K(M_i)$$

  mit selbst gewähltem Klartext $M_1, M_2, \ldots, M_i$

Finde den Schlüssel K oder einen Algorithmus um M_{i+1} aus $C_{i+1} = E_K(M_{i+1})$ abzuleiten.

Ein Chosen–plaintext Angriff kann folgendermaßen aussehen ([64]): Das gängige Verfahren, ein Flugzeug auf einem Radarschirm als Freund oder Feind zu identifizieren, besteht darin, dass die Bodenstation eine zufällige Nachricht (Klartext) an das Flugzeug sendet. Der Transponder des Flugzeugs empfängt diese Message, verschlüsselt sie automatisch[14] und sendet die chiffrierte Nachricht zurück. Die Bodenstation führt mit dem gleichen Schlüssel ebenfalls eine Verschlüsselung dieser zufälligen Nachricht aus. Stellt die Bodenstation fest, dass beide Chiffretexte gleich sind, wird das Flugzeug als freundlich klassifiziert, ansonsten als Feind. Dies nennt man[15] *Identification Friend or Foe, IFF.*

Der Feind kann nun beliebige ausgewählte Nachrichten (Chosen plaintexts) an dieses Flugzeug senden und die resultierenden Chiffretexte untersuchen. Ist der Feind dann in der Lage, aus diesen Klartext/Chiffretextpaaren den Authentifikations–Key zu finden und dann seine eigenen Flugzeuge umzurüsten, kann er seine Flugzeuge als 'freundlich' tarnen.

■ **Chosen–chiphertext Attacke**
Eine Chosen–chiphertext Attacke liegt dann vor, wenn der Angreifer (zeitweise) Zugriff auf den Dechiffrieralgorithmus hat und versucht, mit Hilfe frei gewähltem Chiffretext den zugehörigen Klartext zu gewinnen. Diese Angriffsart wird in den meisten Fällen auf Public–Key Kryptosysteme angewendet.

[14] Das Flugzeug nimmt beim Start den aktuellen Schlüssel mit.
[15] Siehe dazu auch Ross Anderson [2, p. 376] p. 337 und 376.

1.1.5 Das Kerckhoffs Prinzip

Abbildung 1.2: AUGUSTE KERCKHOFFS

In der heutigen Kryptologie wird – in der Regel – streng auf die Einhaltung des sogenannten **Kerckhoffs Prinzips** geachtet. Dieses Prinzip wurde im Jahre 1883 von dem Niederländer AUGUSTE KERCKHOFFS (1835 – 1903)[16] formuliert. KERCKHOFFS publizierte im Jahre 1883 ein für die damalige Zeit richtungsweisendes Buch über Kryptologie mit dem Titel *La cryptographie militaire*, in dem die aktuellen Erkenntnisse über Verschlüsselungen zusammengefaßt wurden. In dieser Arbeit formulierte AUGUSTE KERCKHOFFS sechs Anforderungen an kryptographische Systeme, die auch heute noch durchaus aktuell sind:

① Ein kryptographisches System sollte – wenn nicht theoretisch unbrechbar – dann wenigstens praktisch unbrechbar sein.

② Ist das System kompromittiert, darf dies keine negativen Auswirkungen auf die Kommunikationspartner haben.

③ Der Schlüssel muß leicht merkbar sein – er darf nicht so komplex sein, daß er auf einem Blatt Papier notiert werden muß. Der Schlüssel muß auch leicht änderbar sein.

④ Der Chiffretext sollte einfach und effektiv übertragen werden können.

⑤ Das Equipment zum Ver– und zum Entschlüsseln muß transportabel und durch eine einzige Person bedienbar sein.

⑥ Das kryptographische System muß einfach zu bedienen sein. Die Bedienung darf keine Kenntnis einer langen Liste von Regeln erfordern.

Die zweite Anforderung impliziert das oben formulierte KERCKHOFFSsche Prinzip:

[16] Siehe dazu [118, pp. 230-239]

> Die Sicherheit eines kryptographischen Systems darf niemals von der Geheimhaltung des Algorithmus zur Verschlüsselung abhängen, sondern von der Geheimhaltung eines **Schlüssels**.

Bei dem Design eines kryptographischen Systems muß davon ausgegangen werden, dass es auf die Dauer nicht möglich ist, das Verfahren (*i.e.* den Algorithmus) der Verschlüsselung geheimzuhalten.

1.1.6 Superencryption

Der Begriff **Superencryption** bzw. **Superencipherment** (siehe [150] oder [193, Chap. 15]) steht für die mehrfache Verschlüsselung eines Klartextes. Dabei wird ein Klartext mit einem Kryptosystem verschlüsselt, danach wird der entstandene Chiffretext nochmals chiffriert. Dies kann mit dem gleichen Kryptosystem geschehen, dies kann jedoch auch ein völlig anderes Verschlüsselungssystem sein.

Das Ziel der Superencryption ist der Versuch, die Sicherheit von Kryptosystemen zu erhöhen, ohne den mühsamen Weg gehen zu müssen, einen neuen Algorithmus zu designen. Dabei unterscheidet man unter anderem:

- ☞ *Mehrfache Verschlüsselung*
  Diese Variante bezeichnet das Verfahren, einen Klartext mit (genau) einem Algorithmus mehrfach zu verschlüsseln. Dabei ist ein wesentliches Kriterium, dass entsprechend viele Schlüssel verwendet werden.[17]
- ☞ *Kaskadierung*
  Unter Kaskadierung versteht man in diesem Kontext die mehrfache Verschlüsselung mit unterschiedlichen Kryptosystemen.

[17] Bei diesem Verfahren sind algebraische Betrachtungen des zugrunde liegenden Kryptosystems relevant. Wir werden anhand einiger einfacher klassischer Chifferen sehen, dass eine Erhöhung der Sicherheit durch mehrfache Verschlüsselung nur dann erreicht werden kann, wenn das zugrunde liegende Kryptosystem keine Gruppenstruktur aufweist.

1.2 Verschlüsselungsarten

Die modernen Verschlüsselungsverfahren können in zwei Kategorien eingeteilt werden:

- **Symmetrische Verschlüsselung**
- **Asymmetrische Verschlüsselung**

In diesem Abschnitt betrachten wir die grundlegenden Eigenschaften dieser beiden Verfahren.

Beide Verschlüsselungsarten können als **Block Chiffren** oder Strom Chiffren eingesetzt werden.

☞ **Block Chiffren**
Bei dieser Technik wird ein kompletter Block von Klartextzeichen — in gängigen Verfahren wie der DES Algorithmus zum Beispiel 64 Bit — als Ganzes in einen Chiffretextblock der gleichen Größe transformiert. Diese Methode wird in einer Vielzahl von Algorithmen angewendet, beispielsweise DES, IDEA oder AES oder RSA.

☞ **Strom Chiffren**
Mit Stromchiffren bezeichnet man Verschlüsselungsalgorithmen, die den Klartext Zeichen für Zeichen verschlüsseln und als Output eine Folge von Chiffretextzeichen produzieren. Solche Techniken werden üblicherweise in der mobilen Telekommunikation eingesetzt. Die IEEE 802.11 WLAN Verschlüsselung WEP (Wired Equivalent Privacy) verwendet beispielsweise die Stromchiffre RC4 von Ron Rivest (siehe Abschnitt [4.3]).

1.2.1 Symmetrische Verschlüsselung

In der traditionellen Kryptographie benutzen Sender und Empfänger einer zu verschlüsselnden Nachricht den gleichen geheimen Schlüssel K, der nur diesen beiden Parteien bekannt sein darf. Der Sender der Nachricht benutzt den geheimen Schlüssel, um die Nachricht mit Hilfe eines Verschlüsselungsalgorithmus zu chiffrieren, der Empfänger des Chiffretexts benutzt diesen Schlüssel und einen Entschlüsselungsalgorithmus, um die Nachricht zu dechiffrieren (siehe Abbildung [1.3]). Diese Methode ist unter einer Vielzahl von Begriffen bekannt, darunter

- **Secret Key Verschlüsselung**
- **Symmetrische Verschlüsselung**
- **Private Key Verschlüsselung**

Die Problematik dieses Verfahrens liegt darin, dass die beiden Kommunikationspartner auf sicherem Weg den Schlüssel austauschen müssen, bevor die eigentliche Kommunikation stattfinden kann. Die beiden Kommunikationspartner müssen den geheimen Schlüssel K austauschen, ohne dass eine andere

Partei Zugriff darauf hat. Befinden sich diese Parteien an räumlich getrennten Lokationen, müssen sie einem Courier vertrauen, ein Telefonsystem oder ein anderes vertrauenswürdigem Kommunikationsmedium verwenden, um den Schlüssel im Voraus auf sicherem Wege auszutauschen Jeder, der Zugriff auf diesen Schlüssel hat, kann später sämtliche Nachrichten, die mit diesem Schlüssel chiffriert oder authentifiziert werden, lesen, modifizieren oder fälschen. Die Erzeugung, die sichere Übertragung und die Speicherung von Schlüsseln nennt man **Schlüsselmanagement**. Alle Kryptosysteme — sowohl die symmetrischen als auch asymmetrischen — müssen das Schlüsselmanagement Prozedere lösen. Da sämtliche Schlüssel eines symmetrischen Kryptosystems geheim gehalten werden müssen, haben solche Systeme in der Praxis Probleme, ein sicheres Schlüssel Management zur Verfügung zu stellen. Diese Problematik wird umso deutlicher, je mehr Benutzer dieses System anwenden.

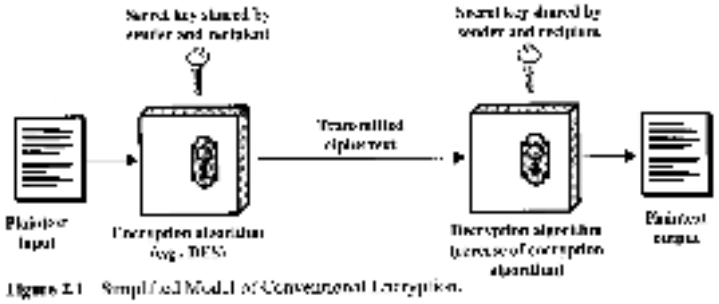


Abbildung 1.3: Ein vereinfachtes Modell der symmetrischen Verschlüsselung.

Die verbreitetsten Einsatzgebiete der symmetrischen Verschlüsselung sind

- **Verschlüsselung**
  Aufgrund der hohen Effizienz — im Vergleich zu asymmterischen Verfahren — heutiger symmetrischer Verschlüsselungsverfahren, werden symmetrische Verfahren genutzt, um Nutzdaten zu chiffrieren.

- **Message Authentication Codes** (MAC)
  In traditionellen Geschäftstransaktionen wird die Gültigkeit eines Vertrags durch eine Unterschrift garantiert. Ein unterschriebener Vertrag dient als legaler Beweis einer Vereinbarung, den der Eigentümer des Vertrags bei Bedarf vor Gericht vorlegen kann. Die Verwendung von Signaturen erfordert die Übertragung und Aufbewahrung geschriebener Verträge.

  Um ein reine digitale Form dieses Papierinstruments zu erhalten, muß jeder Anwender in der Lage sein, eine Nachricht zu generieren, deren Authentizität durch jeden anderen Benutzer *eindeutig* verifiziert werden kann, die jedoch auch durch niemanden sonst — nicht einmal der Empfänger — erzeugt werden kann. Symmetrische Verfahren sind nicht in der Lage, diese Anforderungen für eine *persönliche* Authentifizierung in dieser strengen Form umzusetzen.

Die Authentifizierung einer Nachricht Message Authentication — dies ist auch unter dem Begriff **Digitale Signatur** bekannt — besteht aus zwei Ebenen. Eine Ebene stellt eine Funktion bereit, die einen sogenannten *Authentikator* liefert. Dies ist ein Wert, der dazu benutzt wird, eine Nachricht zu authentifizieren. Dieser Wert wird von Protokollen höherer Ebenen genutzt (das sind sogenannte Authentifizierungs Protokolle); dadurch kann der Empfänger einer Nachricht die Authentizität dieser Nachricht überprüfen.

Falls mit symmetrischen Verfahren Datenintegrität und eine Authentifikation des Ursprungs der Nachricht erzielt werden soll, dann vereinbaren Alice und Bob zunächst einen geheimen Schlüssel k. Alice — die ihre Klartext Nachricht M authentifizieren möchte — berechnet zunächst den sogenannten **Message Authentication Code** (MAC) der Nachricht

$$t = \mathrm{MAC}_k(M)$$

mit Hilfe eines MAC Algorithmus und des symmetrischen Schlüssels k. Alice sendet anschließend die Nachricht M und die Signatur t an Bob. Bob wendet nun den MAC Algorithmus — und den Schlüssel k — auf die empfangene Nachricht M an und berechnet

$$t' = \mathrm{MAC}_k(M)$$

Falls $t = t'$, dann akzeptiert Bob den Sender Alice als Urheberin der Nachricht M.

Der wesentliche Vorteil symmetrischer Kryptosysteme liegt in der hochgradigen Effizienz. Die ausschließliche Verwendung symmetrischer Kryptosysteme führt auf Probleme aufgrund der Architektur solcher Verfahren. Es ist wichtig zu realisieren, dass die Lösung dieser Probleme ein wesentlicher Schritt bedeutet bei der Übertragung traditioneller Geschäftsprozesse auf die Internet Plattform.

☞ **Key Exchange Problem**
Wenn zwei Parteien, die noch niemals zuvor Kontakt miteinander hatten — ein alltägliches Szenario bei geschäftlichen Transaktionen — vertraulich kommunizieren sollen durch Einsatz eines symmetrischen Kryptosystems, dann müssen diese Parteien im Voraus einen Schlüssel vereinbaren, der nur den beiden Parteien bekannt ist.

Mit anderen Worten, das Schlüsselaustauschproblem besteht aus der Bereitstellung eines Übertragungskanals, der sowohl geheim als auch authentisch für die Übertragung des Schlüsselmaterials zwischen den beiden Kommunikationsparteien ist.

☞ **Key Management Problem**
In einem Netzwerk mit n Parteien muss jede Partei unterschiedliches Schlüsselmaterial mit jedem der anderen $n-1$ Parteien vereinbaren. Dies führt auf ein massives Management Problem. Eine (unrealistische) Möglichkeit, dieses Problem zu lösen besteht darin, im Voraus die $n(n-1)/2$ Schlüssel zu erzeugen und zu verteilen. Das sind bei $n = 10.000$ Benutzern immerhin schon 49.995.000 Schlüssel.

☞ **Digitale Signatur**
Da bei symmetrischen Verfahren das Schlüsselmaterial zwischen zwei oder mehreren Parteien verteilt werden muss, können symmetrische Kryptosysteme nicht dazu verwendet werden, verbindliche digitale Signaturverfahren (non-repudiation) zu konstruieren (siehe z.B. [107, p. 4] und [64]).

1.2.2 Asymmetrische Verschlüsselung

Um das Schlüsselmanagement Problem der symmetrischen Verschlüsselung zu lösen, führten im Jahre 1976 **Whitfield Diffie** und **Martin Hellman** ([64]) das Konzept der **Public–Key Kryptographie**[18]. ein.[19]

Abbildung 1.4: Ralph Merkle, Martin Hellman und Whitfield Diffie

Public–Key Kryptosysteme werden für mehrere Zwecke eingesetzt, unter anderem für *Verschlüsselung* und *digitale Signatur*. Das Charakteristische eines Public–Key Kryptosystems ist, dass zum Ver– bzw. Entschlüsseln zwei unterschiedliche Schlüssel verwendet werden. Diese beiden Schlüssel sind *nicht* unabhängig voneinander sondern hängen auf mathematisch sehr subtile Weise voneinander ab.

In einem Public–Key Kryptosystem besitzen die beiden Protagonisten Alice und Bob jeweils ein **Schlüsselpaar**. Einer dieser Schlüssel ist der sogenannte **Public Key** K_{pub}, der andere ist der **Private Key** K_{priv}. Der Public Key wird veröffentlicht — oder kann an den Kommunikationspartner über einen unsicheren Kanal übertragen werden. Der Private Key muß von demjenigen, der das Schlüsselpaar generiert, geheim gehalten werden.

[18] Mittlerweile ist bekannt (siehe zum Beispiel [32, p. 153]), dass einige Jahre vor der Publikation von Diffie und Hellman eine Reihe der in [64] und [182] entwickelten Ideen von James Ellis, Clifford Cocks und Malcolm Williamson vom britischen Geheimdienst **GCHQ** (Government Communications Headquarters) entwickelt wurden. Da diese Entwicklungen per Definition natürlich geheim bleiben mussten, bewirkte erst die Veröffentlichung von Diffie und Hellman einen rasanten Anstieg der Forschung in diesem Gebiet. Die GCHQ Abteilung des britischen Geheimdienstes wurde aus Mitarbeitern des Bletchley Parks nach Ende des 2. Weltkrieges gegründet.

[19] Siehe dazu auch das brilliant geschriebene Buch von Steven Levy [143].

Die Grundvoraussetzung der symmetrischen Verschlüsselung, dass Sender und Empfänger eine geheime Information teilen (den geheimen Schlüssel K) ist eliminiert. Die komplette Kommunikation zwischen Alice und Bob verwendet ausschließlich Public Keys, kein Private Key wird jemals übertragen oder zwischen Alice und Bob ausgetauscht.

Einige der Szenarien, für die Public-Key Kryptosysteme heute eingesetzt werden, sind [132]:

- ☞ Übertragung vertraulicher Nachrichten.
- ☞ *Identifikationssysteme*, *i.e.* Systeme, durch die Benutzer beweisen, dass sie autorisiert sind, um auf bestimmte Daten oder Ressourcen zugreifen zu können, oder dass der Benutzer derjenige ist, der zu sein er behauptet.
- ☞ *Authentifikation*, *i.e.* der Nachweis, dass die Nachricht auch tatsächlich von dem angeblichen Sender stammt und nicht verfälscht wurde.
- ☞ *Verbindlichkeit*, *i.e.* der Schutz gegenüber Personen, die behaupten, etwas nicht zugestimmt zu haben, obwohl sie tatsächlich zugestimmt haben.
- ☞ *Schlüsselaustausch*, *i.e.* zwei Kommunikationspartner, die einen unsicheren Kommunikationskanal verwenden und sich noch nie zuvor begegnet sind, vereinbaren ein geheimen Schlüssel, der nur diesen beiden Parteien bekannt ist, der für ein symmetrisches Kryptosystem verwendet wird.
- ☞ *Electronic Cash*, *i.e.* Mechanismen, die die Anonymität des Zahlers gewährleisten.
- ☞ *Elektronische Wahlen*, *i.e.* Systeme, die garantieren, dass Stimmabgaben vertraulich behandelt und korrekt verbucht werden.

Das grundlegende Szenario der Public–Key Kryptosysteme, die Geheimhaltung (privacy) zu realisieren, läuft in folgenden Schritten ab (siehe Abbildung [1.5]):

① Alice will Bob mit Hilfe eines Public–Key Verfahrens eine Nachricht zukommen lassen, die nur der autorisierte Empfänger — nämlich Bob — lesen kann. Das Ziel ist, die Forderung nach Privacy zu realisieren, ohne dass Alice und Bob zu diesem Zweck ein Geheimnis austauschen müssen.

② Dazu generiert der *Empfänger* der Nachricht — also Bob — ein Schlüsselpaar $(K^B_{\text{pub}}, K^B_{\text{priv}})$.

③ Der Public–Key K^B_{pub} wird von Bob veröffentlicht oder wird von Bob direkt an Alice übermittelt. Der Private–Key K^B_{priv} wird von Bob geheim gehalten, dieser Schlüssel ist nur Bob bekannt.

④ Alice besorgt sich den öffentlichen Schlüssel von Bob (K^B_{pub}) und chiffriert mit diesem Schlüssel die zu übertragende Nachricht.

⑤ Die verschlüsselte Nachricht wird an Bob gesendet.

⑥ Bob dechiffriert nun mit seinem zu K^B_{pub} passenden Private-Key K^B_{priv} das erhaltene Chiffrat und erhält den Klartext. Da nur Bob im Besitz des zu

K^B_{pub} passenden Private Keys ist, kann nur er die verschlüsselte Nachricht dechiffrieren. Damit kann der Sender einer Nachricht sicher sein, daß ausschließlich der autorisierte Empfänger die Nachricht lesen kann.

Damit die Geheimhaltung gewährleistet ist, müssen Public–Key Systeme so aufgebaut sein, dass mit dem öffentlichen Schlüssel K_{pub} des Empfängers vom Sender der Nachricht die Nachricht verschlüsselt werden kann. Die Nachricht kann aber *nur* mit dem passenden — vom Empfänger geheim zu haltenden — private key K_{priv} wieder entschlüsselt werden.[20]

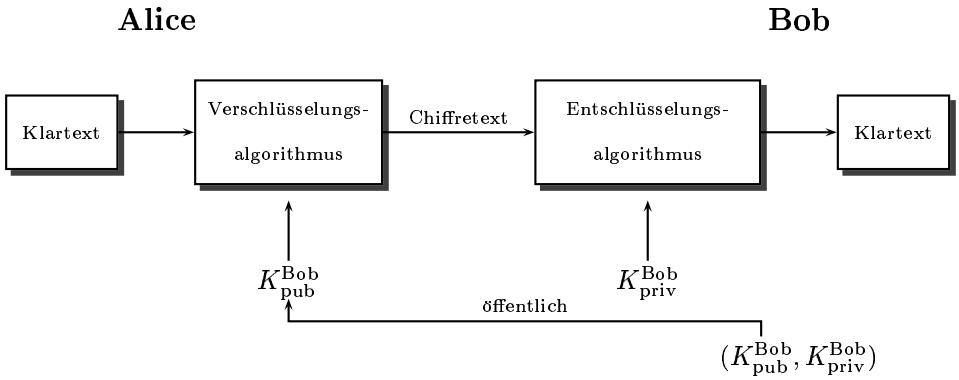


Abbildung 1.5: Vereinfachtes Modell der asymmetrischen Verschlüsselung.

Public–Key Kryptosysteme werden nicht nur für die Verschlüsselung von Klartext verwendet sondern auch für die **Authentifizierung** (Digitale Signaturen) und andere Techniken.

Mit Hilfe von Public–Key Verfahren können Nachrichten prinzipiell wie folgt authentifiziert werden:

① Alice will Bob eine signierte Nachricht zukommen lassen.

② Dazu generiert der *Sender* der Nachricht — also Alice — ein Schlüsselpaar $(K^A_{\text{pub}}, K^A_{\text{priv}})$.

③ Der Public–Key K^A_{pub} wird von Alice veröffentlicht oder dem Empfänger der Nachricht direkt übermittelt. Der Private-Key K^A_{priv} wird von Alice geheim gehalten. Dieser Schlüssel ist nur Alice bekannt und daher identifiziert dieser Schlüssel den Kommunikationspartner Alice.

④ Alice 'verschlüsselt'[21] den Klartext mit ihrem Private-Key K_{priv}. Da nur

[20] In ihrer Arbeit [64] haben DIFFIE und HELLMAN sämtliche Bedingungen an die Schlüssel und das Kryptosystem herausgearbeitet, so dass die sichere Kommunikation zwischen Alice und Bob gewährleistet wird. Sie haben in dieser Arbeit attestieren müssen, dass sie keinen Algorithmus finden konnten, der diese Ideen implementiert.

[21] Es ist evident, dass in diesem Kontext nicht von Verschlüsselung gesprochen werden kann, da jeder, der im Besitz des passenden Public–Keys ist, die Nachricht dechiffrieren kann. Dies ist aber nicht Zweck der Authentifizierung.

sie im Besitz dieses Schlüssels ist, kann nur sie die Urheberin dieser Nachricht sein.

⑤ Alice übermittelt den derart chiffrierten Text an Bob.

⑥ Bob empfängt das Chiffrat und besorgt sich den öffentlichen Schlüssel von Alice K^A_{pub}.

⑦ Bob dechiffriert nun die Nachricht mit dem öffentlichen Schlüssel von Alice. Wenn sinnvoller Klartext entsteht, muss die Nachricht von Alice stammen, da nur sie das Geheimnis K^A_{priv} kennt. Damit kann der Empfänger der Nachricht sicher sein, dass diese auch von dem angeblichen Sender stammt.

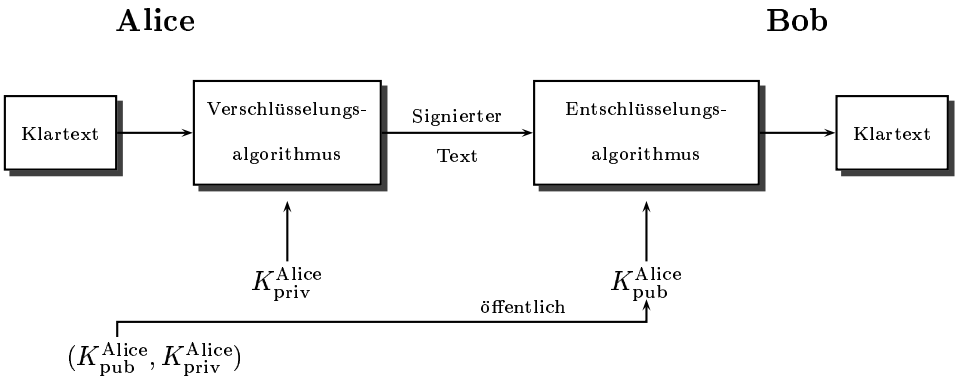


Abbildung 1.6: Authentifizierung mit Public-Key Kryptosystemen.

Anmerkung:

Wie erwähnt hängt in einem Public-Key Kryptosystem der Private–Key in mathematisch subtiler Weise mit dem Public–Key zusammen. Der Grund ist folgender:

Das wesentliche Element von Public–Key Kryptosystemen ist die Verwendung eines Schlüsselpaars (K_{priv}, K_{pub}), wobei der Public Key K_{pub} öffentlich ist. Als Folge dessen hat ein potentieller Angreifer des Systems ebenfalls Zugriff auf den Public Key. Es ist daher immer möglich, ein Public–Key Kryptosystem dadurch zu kompromittieren, dass versucht wird, den Private–Key aus dem öffentlichen Schlüssel abzuleiten. Typischerweise ist die Gegenmaßnahme die, das Problem, den geheimen Schlüssel aus dem öffentlichen Schlüssel abzuleiten, so schwierig wie möglich zu machen. Das mathematische Instrument, dies zu realisieren ist die Verwendung einer **Trapdoor Einwegfunktion**, *i.e.* die autorisierte Benutzung des Systems ist einfach, das nicht–autorisierte Brechen des Systems ist schwierig. Eine Reihe von Public–Key Kryptosysteme sind so gestaltet, dass die Ableitung des Private–Keys aus dem Public–Key die Faktorisierung einer

großen Zahl erfordert. In diesem Fall ist es vom rechnerischen Aufwand her in sinnvoller Zeit nicht möglich, diese Ableitung durchzuführen. Dies ist der Mechanismus, der dem RSA Public–Key Kryptosystem zugrunde liegt.

Public–Key Kryptosysteme sind zum Beispiel:

- Das RSA Verfahren
- Das DIFFIE–HELLMAN Key Exchange Verfahren
- Das ELGAMAL Verfahren
- Knapsack Systeme
- PAILLIER Kryptosysteme
- Elliptische Kurven Kryptosysteme

Weitergehende Diskussionen der Public–Key Kryptosysteme findet man in Kapitel [6].

Um in der Praxis digitale Signaturen zu erzeugen, werden Public–Key Verfahren nicht in der oben dargestellten Reinstform benutzt (siehe z.B. [104, pp. 383 – 405].

1.2.3 Hybridverfahren

Das RSA Verfahren und alle anderen Public–Key Kryptosysteme haben einen entscheidenen Nachteil gegenüber symmetrischen Verfahren. Durch die üblicherweise großen Schlüssellängen und die aufwendigen Rechenoperationen, die Bestandteil der Algorithmen sind, sind sie sehr langsam. Eine RSA Verschlüsselung ist um etwa den Faktor 1000 langsamer als die gleiche Operation mit dem Data Encryption Standard (DES).

RSA und andere Public–Key Kryptosysteme werden aus diesem Grund in der Praxis fast nie verwendet, um komplette Nachrichten zu verschlüsseln. Asymmetrische Verfahren werden stattdessen dazu eingesetzt, einen Schlüssel K für eine symmetrische Verschlüsselung zwischen den beiden Kommunikationspartnern auszutauschen (Session Key). Mit diesem Schlüssel werden dann die eigentlichen Daten, die chiffriert übertragen werden sollen, mit einem symmetrischen Verfahren chiffriert. Daher wird RSA — und andere Public–Key Verfahren — nicht als Verschlüsselungsverfahren sondern als Schlüsselaustauschverfahren eingesetzt. Die Kombination dieser beiden Methoden — symmetrische und asymmetrische Verfahren — nennt man **Hybridverfahren**.

Das grundlegende Protokoll zur Implementierung von Hybridsystemen hat folgenden Ablauf:

① Bob sendet Alice seinen öffentlichen Schlüssel.

② Alice generiert einen zufälligen Sitzungsschlüssel K, verschlüsselt diesen mit Bobs Public Key und sendet das Chiffrat an Alice.

$$E_B(K)$$

③ Bob entschlüsselt das Chiffrat mit seinem Private Key um den Session Key zu erhalten.

$$D_B(E_B(K)) = K$$

④ Alice und Bob verschlüsseln ab diesem Zeitpunkt ihre Nachrichten mit einem symmetrischen Kryptosystem mit dem auf diese Weise ausgetauschten Session Key K.

1.3 Who is who in der Kryptographie

☞ **Das American National Standards Institute (ANSI)**
Das *American National Standards Institute* (ANSI) ist eine private non-profit Organisation in den USA, deren Aufgabe die Entwicklung von Standards in vielen technischen Bereichen ist. Die ANSI hat in den USA die gleiche Funktion wie das DIN Institut in Deutschland.

Das *Accredited Standards Committee X9* (ASC X9)[22] ist eine Arbeitsgruppe des ANSI, welches den Auftrag hat, für die Finanzdienstleistungsindustrie Standards zu entwickeln, festzulegen, zu unterhalten und voranzutreiben. Ziel dabei ist, Finanzdienstleistungen und deren Produkte zu ermöglichen bzw. zu erleichtern. Unter diesem Auftrag erfüllt das ASC X9 Kommitteedie folgenden Ziele

- Unterstützung bestehender Standards
- Förderung der Entwicklung neuer, offener Standards, die auf allgemeinem Konsens basieren
- Bereitstellung eines gemeinsamen Grundlage für alle Standards, die die Finanzdienstleistungsindustrie betreffen
- Akzente setzen auf aktuelle und zukünftige Anforderungen an Standards der Finanzleistungsindustrie
- Vorantreiben und Verbreitung der Standards
- Teilnahme an der Entwicklung internationaler Standards (ISO)

Das ANSI X9 Kommittee hat eine Vielzahl von kryptologischer Standards für die Finanzwelt adaptiert, beispielsweise in dem Dokument ANSI X9.55 (1997)[23] mit dem Titel:

> *Public Key Cryptography for the Financial Services Industry: Extensions to Public Key Certificates and Certificate Revocation Lists*

☞ **Die NSA**
Die National Security Agency (NSA) gilt als der größte und wahrscheinlich finanziell am besten ausgestattete Nachrichtendienst der Welt[24]. Die NSA ist für die weltweite Überwachung und Entschlüsselung von elektronischer Kommunikation zuständig und in dieser Funktion ein Teil der Intelligence Community, in der sämtliche Nachrichtendienste der USA zusammengefasst sind. Die NSA ist administrativ und technisch dem amerikanischen Verteidigungsministerium unterstellt, operativ aber direkt dem

[22] Siehe die URL:

http://www.x9.org/home/

[23] Siehe beispielsweise die URL:

http://webstore.ansi.org/ansidocstore/find.asp?.

[24] Siehe die detaillierte Beschreibung der NSA in dem Buch von James Bamford in [6].

National Security Advisor, zur Zeit STEPHEN HADLEY, den es bei seiner Entscheidungsfindung mit nachrichtendienstlichen Erkenntnissen unterstützt.

Zur Überwachung der weltweiten Kommunikation unterhält die NSA zu-

Abbildung 1.7: Die ehemalige Echelon Abhöranlage der NSA bei Bad Aibling, die im Jahre 2004 geschlossen wurde.

sammen mit anderen staatlichen Organisationen eine Reihe von Abhöranlagen, die unter dem Namen *Echelon* bekannt sind. Bis 2004 betrieb die NSA in Bad Aibling eine Anlage (siehe Abbildung [1.7]). Die dort stationierten Nachrichteneinheiten wurden nach England, in die Türkei und in die Nähe von Darmstadt (Griesheim) versetzt.

☞ **Das Government Communications Headquarters, GCHQ**
Der britische Geheimdienst unterhält eine[25]

☞ **Das NIST**
Das **National Institute of Standards and Technology** – früher NBS National Bureau of Standards – ist eine Abteilung des amerikanischen Departments of Commerce und unterhält eine Vielzahl an Abteilungen zu Entwicklung von Standards.

Eine unter acht Divisionen des NIST – Information Technology Labs ist das COMPUTER SECURITY RESOURCE CENTER[26] (CSRC)

☞ **Die International Organization for Standardization (ISO)**
Die ISO publiziert weltweite Standards in vielen Bereichen, darunter auch kryptographische Protokolle.

☞ **Das IEEE**
Die **Institute of Electrical and Electronics Engineers** Untergruppe mit Bezeichnung IEEE P1363[27] entwickelt Standards für die Public Key Kryptographie auf einer sehr breiten Basis.

[25] Siehe

http://www.gchq.gov.uk/about/index.html

[26] Siehe die URL:

http://csrc.nist.gov

[27] Siehe die URL:

- **Die Electronic Frontier Foundation (EFF)**
  Die **Electronic Frontier Foundation**

- Das NESSIE Projekt
  Das Akronym **NESSIE** steht für *New European Schemes for Signatures, Integrity, and Encryption* und steht für ein von der EU finanziertes Projekt mit dem Ziel, kryptographisch starke Primitive zu entwickeln[28].

- **Bruce Schneier, Counterpane**
  Bruce Schneier ist einer der führenden, weltweit anerkannten Experten in der Sicherheitstechnologie, Autor mehrerer Bücher dieses Themas [81, 194, 204] sowie Architekt mehrerer, erfolgreicher Kryptosysteme wie Blowfish oder Twofish.

Abbildung 1.8: Bruce Schneier

- **Ron Rivest**
  Ron Rivest ist Professor am MIT Department of Electrical Engineering and Computer Science.

- **Adi Shamir**
  Einer der herausragenden Kryptologen

- **RSA Laboratories**
  Die RSA Labs ist die Forschungsabteilung der Firma RSA Security.

- **RSA Security**
  Die Firma RSA Security

- **Das Bundesamt für Sicherheit in der Informationstechnologie, BSI**
  Das Bundesamt für Sicherheit in der Informationstechnologie, BSI, wurde im Jahre 1991 gegründet. Es ging aus der Zentralstelle fr Sicherheit in der Informationstechnik (ZSI) hervor, deren Vorgängerbehörde die dem

http://grouper.ieee.org/groups/1363

[28] Siehe die Site:

https://www.cosic.esat.kuleuven.ac.be/nessie

Bundesnachrichtendienst (BND) unterstellte Zentralstelle fr das Chiffrierwesen (ZfCh) war. Der Mathematiker Dr. Otto Leiberich, seit 1957 beim BND und dort zuletzt Leiter der ZfCh, war erster Prsident des BSI.

Das BSI ist die zentrale Zertifizierungsstelle für die Sicherheit von IT-Systemen in Deutschland (Computer- und Datensicherheit, Datenschutz). Prüfung und Zertifizierung ist möglich in Bezug auf die Standards des IT-Grundschutzhandbuch, dem Grünbuch, ITSEC und den Common Criteria.

☞ **Phil Zimmermann**
Phil Zimmermann war der erste, der die Public Key-Kryptographie

Abbildung 1.9: Phil Zimmermann

als Software der Allgemeinheit leicht zugänglich machte. Dies führte dazu, dass die US-Zollbehörden ihn zum Ziel einer drei Jahre dauernden Untersuchung machten, da die Regierung der Ansicht war, dass US amerikanische Exportbeschränkungen für kryptographische Software verletzt worden waren, als PGP ab 1991 nach Veröffentlichung als Freeware im Internet seinen Siegeszug rund um die Welt antrat. Nachdem die Regierung den Fall Anfang 1996 ohne Anklage hatte fallen lassen, gründete Zimmermann die Firma PGP Inc., die im Dezember 1997 von Network Associates Inc. (NAI) übernommen wurde. Dort blieb er drei Jahre lang als Senior Fellow. Im Jahr 2002 wurde PGP der NAI von einer neuen Firma namens PGP Corporation abgekauft, wo Zimmermann nun als spezieller Berater tätig ist. Zimmermann ist ferner ein 'Fellow' des Stanford Law School's Center for Internet and Society.

Im März 2006 stellte er die Beta-Version einer neu entwickelten Software namens Zfone zur Verschlüsselung von VoIP-Telefonaten vor.

☞ **David Chaum**
David Chaum entwickelte in den 80er Jahren die kryptogrpahischen Protokolle

1.4 Software

✗ `CrypTool`

✗ `Truecrypt`
Das Truecrypt Programm erstellt verschlüsselte Datencontainer, in die sich beliebige Dateien verschieben lassen. `http://www.truecrypt.org`

✗ `Steganos Safe One`

✗ `Pretty Good Privacy`

1.5 Review Questions

1. Was sind die wesentlichen Bestandteile eines symmetrischen Kryptosystems?
2. Was sind die wesentlichen Bestandteile eines asymmetrischen Kryptosystems?
3. Was sind die beiden grundlegenden Operationen, die in heutigen symmetrischen Verschlüsselungsalgorithmen eingesetzt werden?
4. Was ist der Unterschied zwischen einer Block Chiffre und einer Strom Chiffre?
5. Was ist Steganographie?
6. Was sind die fünf wichtigsten Ziele der Kryptographie?
7. Was ist der Unterschied zwischen Authentifikation und digitaler Signatur?
8. Was bedeutet Autorisierung?
9. Was ist die mathematische Definition nach SHANNON eines Kryptosystems?
10. Welche drei Kategorien an Sicherheit für Kryptosysteme gibt es?
11. Welche Angriffsarten auf Kryptosysteme kennen Sie?
12. Was bedeutet IFF?
13. Was ist die Aussage des KERCKHOFFS Prinzips?
14. Was bedeutet Superencryption? Was sind Hybrid Verfahren?

Kapitel 2

Klassische Kryptosysteme

Die in diesem Kapitel betrachteten Kryptosysteme sind ausnahmslos symmetrische Verschlüsselungsverfahren. Wir sehen uns diese Verfahren hier mit dem Hintergedanken an, die grundlegenden Eigenschaften und Verfahren der Kryptologie an relativ einfachen und nachvollziehbaren Verfahren zu studieren. Diese Methoden haben auch in heuten Verfahren nach wie vor Relevanz. Moderne symmetrische Verschlüsselungsmethoden sind in der Regel sehr komplex, so dass die wesentlichen Punkte zweckmäßiger an einfacher zu handhabenden Chiffren betrachtet werden. Die hier diskutierten Verfahren findet man in einer Vielzahl an Büchern, siehe insbesondere [213, 215].

Wir werden in diesem Skript die folgenden, allgemein akzeptierten Konventionen anwenden.

① Klartext wird in Kleinbuchstaben geschrieben.

② Chiffretext wird in Großbuchstaben geschrieben.

③ Im Klartext werden Leerzeichen und Interpunktionszeichen weggelassen. Der Grund liegt darin, dass die Verschlüsselung dieser Klartextzeichen einen möglichen Angriffspunkt für die Kryptanalyse liefert.

④ Die Zeichen des Klartext bzw. Chiffretext Alphabets werden durch Zahlen codiert, beginnend mit 0.

```
a = A = 0    j = J = 9    s = S = 18
b = B = 1    k = K = 10   t = T = 19
c = C = 2    l = L = 11   u = U = 20
d = D = 3    m = M = 12   v = V = 21
e = E = 4    n = N = 13   w = W = 22
f = F = 5    o = O = 14   x = X = 23
g = G = 6    p = P = 15   y = Y = 24
h = H = 7    q = Q = 16   z = Z = 25
i = I = 8    r = R = 17
```

2.1 Skytale

Das älteste überlieferte Verschlüsselungsverfahren [118, p. 82] ist die **Skytale**, das die Spartaner bereits 500 vor Chr. für militärische Zwecke einsetzten. Die Skytale ist ein Holzstab, um den ein Streifen Leder oder Papyrus gewickelt wird.

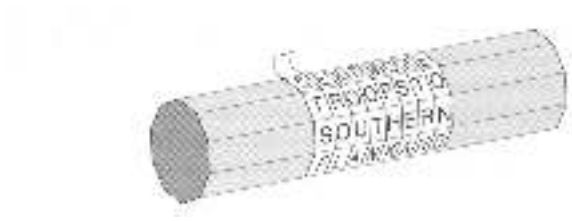


Abbildung 2.1: Wird der Lederstreifen vom Holzstab (Skytale) des Absenders gelöst, scheint er mit einer willkürlichen Reihe von Buchstaben beschrieben, S,T,S,F,... Nur dann, wenn der Streifen um eine andere Skytale mit dem richtigen Durchmesser gewickelt wird, dann erscheint wieder die Nachricht [204].

Der Sender der Nachricht schreibt seine Nachricht der Länge des Stabes nach auf den Streifen und wickelt ihn dann ab. Ohne um den Holzstab (mit dem richtigen Durchmesser) gewickelt, scheint der Leder-/Papyrusstreifen nur eine sinnlose Anreihung von Buchstaben zu enthalten. Mit anderen Worten, die Buchstaben des Klartextes sind von der Form erhalten geblieben, die Position hat sich aber geändert. Daher handelt es sich bei der Skytale um eine **Transpositionschiffre**. Durch einen Boten wurde der Leder-/Papyrusstrifen vom Sender zum Empfänger überbracht.

Damit der Empfänger den verschlüsselten Text wird dechiffrieren und lesen kann, benötigt er einen Holzstab mit dem gleiche Durchmesser wie der Sender. Der Durchmesser des Holzstabes ist also der Schlüssel dieses Kryptosystems. Siehe auch das Buch von KRIEGER [134], pp. 63ff.

2.2 Shift Chiffre

Die Shift Chiffre ist ein Kryptosystem über dem Klartextalphabet $\{M\} = \mathbb{Z}_{26}$, Chiffretextalphabet $\{C\} = \mathbb{Z}_{26}$ dem Schlüsselraum $\{K\} = \mathbb{Z}_{26}$, das folgendermaßen definiert ist.[1]

Definition [2.3]:

Sei $\{M\} = \{C\} = \{K\} = \mathbb{Z}_{26}$. Für $0 \leq k \leq 25$ (k - Schlüssel) definieren wir

$$\begin{array}{rccll} \text{CAESAR}_k: & \mathbb{Z}_{26} & \longrightarrow & \mathbb{Z}_{26} & \\ & x & \longmapsto & y & = \text{CAESAR}_k(x) \\ & & & & = (x+k) \bmod 26, \end{array}$$

und der Dechiffrierung

$$\begin{array}{rccll} (\text{CAESAR}_k)^{-1}: & \mathbb{Z}_{26} & \longrightarrow & \mathbb{Z}_{26} & \\ & y & \longmapsto & x & = (\text{CAESAR}_k)^{-1}(y) \\ & & & & = (y-k) \bmod 26 \\ & & & & = \text{CAESAR}_{(-k)}(y). \end{array}$$

Bemerkungen:

1. In der Literatur wird üblicherweise der Fall $k = 3$ als CÄSAR Chiffre bezeichnet.

2. Es ist sehr lehrreich, zu zeigen, dass CAESAR_k und $\text{CAESAR}_{(-k)}$ invers zueinander sind, *i.e.* die Verschlüsselung und die anschließende Entschlüsselung müssen den ursprünglichen Klartext ergeben. Sei also $x \in \{M\}$, *i.e.* ein beliebiges Klartextzeichen. Dann folgt:

$$\begin{aligned} \big[\text{CAESAR}_k \circ \text{CAESAR}_{(-k)}\big](x) &= \text{CAESAR}_k\Big(\text{CAESAR}_{(-k)}(x)\Big) \\ &= \text{CAESAR}_k\Big(\underbrace{(x-k) \bmod 26}_{\in \mathbb{Z}_{26}}\Big) \\ &= \Big[(x-k) \bmod 26 + k\Big] \bmod 26 \\ &= \Big[(x-k) \bmod 26 + k \bmod 26\Big] \bmod 26 \\ &= (x-k+k) \bmod 26 \\ &= x \bmod 26 \\ &= x \qquad \text{da } x \in \mathbb{Z}_{26}. \end{aligned}$$

In Schritt 3 benutzen wir die Tatsache, dass der Schlüssel eine Zahl zwi-

[1] Siehe HOLDEN [113], p. 5 oder BAUER [11].

schen 0 und 25 ist, *i.e.* $k \in \mathbb{Z}_{26}$. Daher gilt

$$k = k \bmod 26.$$

In Schritt 4 machen wir Gebrauch von der modularen Additionsregel

$$(a \bmod n + b \bmod n) \bmod n = (a + b) \bmod n.$$

Dieses Ergebnis zeigt, dass die obigen Abbildungen invers zueinander sind.

Beispiel [2.1]

Alice will eine Nachricht an Bob senden, die mit der Shift Chiffre verschlüsselt wird. Alice und Bob tauschen zuvor auf sicherem Weg den — nur Alice und Bob bekannten — Schlüssel $k = 11$ aus.

Der zu verschlüsselnde Klartext lautet:

`berufsakademie`

✎ Codierung
Im ersten Schritt codiert Alice den Klartext in eine Ziffernfolge gemäß der konventionnellen Zuordnung.

b	e	r	u	f	s	a	k	a	d	e	m	i	e
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
1	4	17	20	5	18	0	10	0	3	4	12	8	4

✎ Verschlüsselung
Jede Ziffer x wird nun nach dem Verschlüsselungsalgorithmus

$$x \longrightarrow (x + k) \bmod 26 = (x + 11) \bmod 26$$

chiffriert.

1	$\longrightarrow$	$(1 + 11) \bmod 26 = 12 \bmod 26$	$\longrightarrow$	M
4	$\longrightarrow$	$(4 + 11) \bmod 26 = 15 \bmod 26$	$\longrightarrow$	P
17	$\longrightarrow$	$(17 + 11) \bmod 26 = 28 \bmod 26 = 2 \bmod 26$	$\longrightarrow$	C
20	$\longrightarrow$	$(20 + 11) \bmod 26 = 31 \bmod 26 = 5 \bmod 26$	$\longrightarrow$	F
5	$\longrightarrow$	$(5 + 11) \bmod 26 = 16 \bmod 26$	$\longrightarrow$	Q
18	$\longrightarrow$	$(18 + 11) \bmod 26 = 29 \bmod 26 = 3 \bmod 26$	$\longrightarrow$	D
0	$\longrightarrow$	$(0 + 11) \bmod 26 = 11 \bmod 26$	$\longrightarrow$	L
10	$\longrightarrow$	$(10 + 11) \bmod 26 = 21 \bmod 26$	$\longrightarrow$	V
0	$\longrightarrow$	$(0 + 11) \bmod 26 = 11 \bmod 26$	$\longrightarrow$	L
3	$\longrightarrow$	$(3 + 11) \bmod 26 = 14 \bmod 26$	$\longrightarrow$	O
4	$\longrightarrow$	$(4 + 11) \bmod 26 = 15 \bmod 26$	$\longrightarrow$	P
12	$\longrightarrow$	$(12 + 11) \bmod 26 = 23 \bmod 26$	$\longrightarrow$	X
8	$\longrightarrow$	$(8 + 11) \bmod 26 = 19 \bmod 26$	$\longrightarrow$	T
4	$\longrightarrow$	$(4 + 11) \bmod 26 = 15 \bmod 26$	$\longrightarrow$	P

✎ Der resultierende Chiffretext

```
MPCFQDLVLOPXTP
```

wird an Bob übertragen.

Um den erhaltenen Chiffretext zu dechiffrieren, verfährt Bob folgendermaßen:

- Codierung
  Bob codiert den erhaltenen Chiffretext in eine Ziffernfolge gemäß der vereinbarten Zuordnung:

M	P	C	F	Q	D	L	V	L	O	P	X	T	P
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
12	15	2	5	16	3	11	21	11	14	15	23	19	15

- Dechiffrierung
  Jede Ziffer X wird nun nach dem Entschlüsselungsalgorithmus

$$X \longrightarrow (X - k) \bmod 26 = (X - 11) \bmod 26$$

  dechiffriert.

Anmerkungen

❶ Da durch die Shift Chiffre jeder Klartextbuchstabe auf genau einen Chiffretextbuchstaben abgebildet wird, wird das Klartextalphabet auf genau ein Chiffrealphabet abgebildet. Das bedeutet, dass die Shift Chiffre eine **monoalphabetische Chiffre** ist.

❷ Das oben durchgespielte Beispiel zeigt bereits eine charakteristische Eigenschaft der Shift Chiffre — letztendlich ist es eine Eigenschaft jeder monoalphabetischen Chiffre. Im Klartext `berufsakademie` kommt der Buchstabe `e` dreimal vor, an den Positionen 2, 11 und 14. Besieht man sich den zugehörigen Chiffretext, dann erkannt man an den entsprechenden Positionen den Buchstaben `P`. Die statistischen Eigenschaften des Klartexts werden auf den Chiffretext übertragen.

❸ Die Shift Chiffre zählt zu der Klasse der **Substitutions Chiffren**. Der Grund liegt darin, dass durch das Verschlüsselungsverfahren die Klartextzeichen durch andere Chiffretextzeichen ersetzt (substituiert) werden.

❹ Der Schlüsselraum der Shift Ciffre ist $\{K\} = \mathbb{Z}_{26}$ mit $\mid \{K\} \mid = 26$, *i.e.* es gibt 26 verschiedene Schlüssel.

❺ Die Sicherheit der Verschlüsselung erhöht sich nicht, wenn man zweimal hintereinander mit zwei verschiedenen Schlüsseln K_1 und K_2 verschlüsselt. Da der Schlüssel K_1 eine Verschiebung der Buchstaben des Alphabets um K_1 Stellen und der Schlüssel K_2 um K_2 Stellen bedeutet, ist die Hintereinanderausführung zweier Verschlüsselungen eine Verschiebung um $K_1 + K_2$ Stellen. Da $K_1 + K_2$ wieder eine Zahl K aus $\mathbb{Z}_{26}$ ist, gilt $K_1 + K_2 = K$. Dies ist letztlich die Gruppenstruktur der Shift–Chiffre unter Hintereinanderausführung.

2.3 Die Affine Chiffre

Die affine Chiffre ist ein Kryptosystem über dem Klartextalphabet $\{M\} = \mathbb{Z}_{26}$, Chiffrealphabet $\{C\} = \mathbb{Z}_{26}$ und dem Schlüsselraum $\{K\}$, das folgendermaßen definiert ist.

Definition [2.4]:

Sei $\{M\} = \{C\} = \mathbb{Z}_{26}$ und

$$\{K\} = \{(\alpha, \beta) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26} \mid \text{ ggT}(\alpha, 26) = 1\},$$

dann ist

$$\begin{aligned} A_{(\alpha,\beta)} &: \mathbb{Z}_{26} \longrightarrow \mathbb{Z}_{26} \\ x &\longmapsto y = A_{(\alpha,\beta)}(x) = (\alpha x + \beta) \bmod 26 \\ &\quad \text{mit } \alpha, \beta \in \mathbb{Z}_{26} \text{ und } \text{ggT}(\alpha, 26) = 1 \end{aligned}$$

die affine Chiffre. Die Dechiffrierung ist:

$$\begin{aligned} (A_{(\alpha,\beta)})^{-1} &: \mathbb{Z}_{26} \longrightarrow \mathbb{Z}_{26} \\ y &\longmapsto (A_{(\alpha,\beta)})^{-1} y = (\alpha^{-1}(x - \beta)) \bmod 26 \\ &\quad \text{wobei } \alpha^{-1} \cdot \alpha \equiv 1 \bmod 26 \end{aligned}$$

Beispiel [2.2]

Alice will eine Nachricht an Bob senden, die mit der affinen Chiffre verschlüsselt wird. Da die affine Chiffre ein symmetrisches Verfahren ist, müssen Alice und Bob zuvor auf sicherem Weg den Schlüssel austauschen. Das bedeutet in diesem Fall, sie müssen ein Zahlenpaar (α, β) wählen, wobei α so zu wählen ist, dass die Bedingung $\text{ggT}(\alpha, 26) = 1$ erfüllt ist. Alice und Bob einigen sich auf $(5, 12)$, dies erfüllt die Bedingung $\text{ggT}(5, 26) = 1$.

Der zu verschlüsselnde Klartext lautet:

`wirtschaftsinformatik`

✎ Codierung
Im ersten Schritt codiert Alice den Klartext in eine Ziffernfolge gemäß der konventionellen Zuordnung.

w	i	r	t	s	c	h	a	f	t	s	i	n	f	o	r	m	a	t	i	k
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
22	8	17	19	18	2	7	0	5	19	18	8	13	5	14	17	12	0	19	8	10

✎ Verschlüsselung
Jede dieser Ziffern x wird nun nach dem Verschlüsselungsalgorithmus

$$x \longrightarrow (\alpha \cdot x + \beta) \bmod 26 = (5 \cdot x + 12) \bmod 26$$

chiffriert.

$$
\begin{array}{rcll}
22 & \longrightarrow & (22 \cdot 5 + 12) \bmod 26 = 18 \bmod 26 & \longrightarrow \quad \texttt{S} \\
8 & \longrightarrow & (8 \cdot 5 + 12) \bmod 26 = 0 \bmod 26 & \longrightarrow \quad \texttt{A} \\
17 & \longrightarrow & (17 \cdot 5 + 12) \bmod 26 = 19 \bmod 26 & \longrightarrow \quad \texttt{T} \\
19 & \longrightarrow & (19 \cdot 5 + 12) \bmod 26 = 3 \bmod 26 & \longrightarrow \quad \texttt{D} \\
18 & \longrightarrow & (18 \cdot 5 + 12) \bmod 26 = 24 \bmod 26 & \longrightarrow \quad \texttt{Y} \\
2 & \longrightarrow & (2 \cdot 5 + 12) \bmod 26 = 22 \bmod 26 & \longrightarrow \quad \texttt{W} \\
7 & \longrightarrow & (7 \cdot 5 + 12) \bmod 26 = 21 \bmod 26 & \longrightarrow \quad \texttt{V} \\
0 & \longrightarrow & (0 \cdot 5 + 12) \bmod 26 = 12 \bmod 26 & \longrightarrow \quad \texttt{M} \\
5 & \longrightarrow & (5 \cdot 5 + 12) \bmod 26 = 11 \bmod 26 & \longrightarrow \quad \texttt{L} \\
19 & \longrightarrow & (19 \cdot 5 + 12) \bmod 26 = 3 \bmod 26 & \longrightarrow \quad \texttt{D} \\
18 & \longrightarrow & (18 \cdot 5 + 12) \bmod 26 = 24 \bmod 26 & \longrightarrow \quad \texttt{Y} \\
8 & \longrightarrow & (8 \cdot 5 + 12) \bmod 26 = 0 \bmod 26 & \longrightarrow \quad \texttt{A} \\
13 & \longrightarrow & (13 \cdot 5 + 12) \bmod 26 = 19 \bmod 26 & \longrightarrow \quad \texttt{Z} \\
5 & \longrightarrow & (5 \cdot 5 + 12) \bmod 26 = 11 \bmod 26 & \longrightarrow \quad \texttt{L} \\
14 & \longrightarrow & (14 \cdot 5 + 12) \bmod 26 = 4 \bmod 26 & \longrightarrow \quad \texttt{E} \\
17 & \longrightarrow & (17 \cdot 5 + 12) \bmod 26 = 19 \bmod 26 & \longrightarrow \quad \texttt{T} \\
12 & \longrightarrow & (12 \cdot 5 + 12) \bmod 26 = 20 \bmod 26 & \longrightarrow \quad \texttt{U} \\
0 & \longrightarrow & (0 \cdot 5 + 12) \bmod 26 = 12 \bmod 26 & \longrightarrow \quad \texttt{M} \\
19 & \longrightarrow & (19 \cdot 5 + 12) \bmod 26 = 3 \bmod 26 & \longrightarrow \quad \texttt{D} \\
8 & \longrightarrow & (8 \cdot 5 + 12) \bmod 26 = 0 \bmod 26 & \longrightarrow \quad \texttt{A} \\
10 & \longrightarrow & (10 \cdot 5 + 12) \bmod 26 = 10 \bmod 26 & \longrightarrow \quad \texttt{K}
\end{array}
$$

✎ Der resultierende Chiffretext ist also

$$\texttt{SATDYWVMLDYAZLETUMDAK}$$

Dieser wird an Bob übertragen.

✎ Bob muß nun zuerst die Dechiffrierabbildung konstruieren. Die Dechiffrierung ist:

$$
\begin{aligned}
(A_{(\alpha,\beta)})^{-1} \; &: \mathbb{Z}_{26} \longrightarrow \mathbb{Z}_{26} \\
y \; &\longmapsto \; (A_{(\alpha,\beta)})^{-1}(y) = (\alpha^{-1}(y-\beta)) \bmod 26 \\
&\qquad \text{wobei } \alpha^{-1} \cdot \alpha \equiv 1 \bmod 26.
\end{aligned}
$$

Für den verwendeten Schlüssel ist

$$(A_{(5,12)})^{-1}(y) = 5^{-1}(y-12) \bmod 26,$$

mit

$$5^{-1} \cdot 5 \equiv 1 \bmod 26.$$

Mit anderen Worten, Bob hat das multiplikative Inverse zu 5 modulo 26 zu bestimmen. Im aktuellen Fall ist dies noch recht einfach (z.B. durch Ausprobieren) zu finden, da

$$105 = 21 \cdot 5 = 4 \cdot 26 + 1 \equiv 1 \bmod 26.$$

Damit haben wir eine Zahl gefunden (*i.e.* 21) mit der Eigenschaft

$$5 \cdot 21 \equiv 1 \bmod 26.$$

Das heisst, 21 ist das **multiplikative Inverse** von 5 mod 26. Damit lautet die Dechiffrierabbildung:

$$(A_{(5,12)})^{-1}(y) = 21 \cdot (y - 12) \bmod 26 = (21 \cdot y + 8) \bmod 26.$$

- Mit dieser Dechiffrierabbildung wird nun der erhaltene Chiffretxt folgendermaßen codiert und entschlüsselt:

S	$\longrightarrow$	18	$\longrightarrow$	$(21 \cdot 18 + 8) \bmod 26 = 22 \bmod 26$	$\longrightarrow$	w
A	$\longrightarrow$	0	$\longrightarrow$	$(21 \cdot 0 + 8) \bmod 26 = 8 \bmod 26$	$\longrightarrow$	i
T	$\longrightarrow$	19	$\longrightarrow$	$(21 \cdot 19 + 8) \bmod 26 = 17 \bmod 26$	$\longrightarrow$	r
D	$\longrightarrow$	3	$\longrightarrow$	$(21 \cdot 3 + 8) \bmod 26 = 19 \bmod 26$	$\longrightarrow$	t
Y	$\longrightarrow$	24	$\longrightarrow$	$(21 \cdot 24 + 8) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	s
W	$\longrightarrow$	22	$\longrightarrow$	$(21 \cdot 22 + 8) \bmod 26 = 2 \bmod 26$	$\longrightarrow$	c
V	$\longrightarrow$	23	$\longrightarrow$	$(21 \cdot 23 + 8) \bmod 26 = 7 \bmod 26$	$\longrightarrow$	h
M	$\longrightarrow$	12	$\longrightarrow$	$(21 \cdot 12 + 8) \bmod 26 = 0 \bmod 26$	$\longrightarrow$	a
L	$\longrightarrow$	11	$\longrightarrow$	$(21 \cdot 11 + 8) \bmod 26 = 5 \bmod 26$	$\longrightarrow$	f
D	$\longrightarrow$	3	$\longrightarrow$	$(21 \cdot 3 + 8) \bmod 26 = 19 \bmod 26$	$\longrightarrow$	t
Y	$\longrightarrow$	24	$\longrightarrow$	$(21 \cdot 24 + 8) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	s
A	$\longrightarrow$	0	$\longrightarrow$	$(21 \cdot 0 + 8) \bmod 26 = 8 \bmod 26$	$\longrightarrow$	i
Z	$\longrightarrow$	25	$\longrightarrow$	$(21 \cdot 25 + 8) \bmod 26 = 13 \bmod 26$	$\longrightarrow$	n
L	$\longrightarrow$	11	$\longrightarrow$	$(21 \cdot 11 + 8) \bmod 26 = 5 \bmod 26$	$\longrightarrow$	f
E	$\longrightarrow$	5	$\longrightarrow$	$(21 \cdot 5 + 8) \bmod 26 = 14 \bmod 26$	$\longrightarrow$	o
T	$\longrightarrow$	19	$\longrightarrow$	$(21 \cdot 19 + 8) \bmod 26 = 17 \bmod 26$	$\longrightarrow$	r
U	$\longrightarrow$	20	$\longrightarrow$	$(21 \cdot 20 + 8) \bmod 26 = 12 \bmod 26$	$\longrightarrow$	m
M	$\longrightarrow$	12	$\longrightarrow$	$(21 \cdot 12 + 8) \bmod 26 = 0 \bmod 26$	$\longrightarrow$	a
D	$\longrightarrow$	3	$\longrightarrow$	$(21 \cdot 3 + 8) \bmod 26 = 19 \bmod 26$	$\longrightarrow$	t
A	$\longrightarrow$	0	$\longrightarrow$	$(21 \cdot 0 + 8) \bmod 26 = 8 \bmod 26$	$\longrightarrow$	i
K	$\longrightarrow$	10	$\longrightarrow$	$(21 \cdot 10 + 0) \bmod 26 = 10 \bmod 26$	$\longrightarrow$	k

Dies führt also auf den korrekten Klartext

`wirtschaftsinformatik`

Anmerkungen:

❶ Die Shift Chiffre ist die affine Chiffre mit dem Parameter $(1, \beta)$, *i.e.*

$$\text{CAESAR}_k = A_{(1,k)}.$$

❷ Die Anzahl der möglichen Schlüssel der affinen Chiffre, *i.e.* die Größe des Schlüsselraums

$$\{K\} = \{(\alpha, \beta) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26} \mid \text{ggT}(\alpha, 26) = 1\}$$

ist: $\mid \{K\} \mid = 312$.

Dies lässt sich folgendermaßen einsehen. Für den Parameter $\beta \in \mathbb{Z}_{26}$ sind 26 verschiedene Werte zulässig. Für den anderen Parameter α stellt sich

die Frage, wieviele Werte $0\alpha < 26$ existieren mit $\text{ggT}(\alpha, 26) = 1$. Oder: Wieviele Zahlen < 26 existieren, die coprim zu 26 sind. Die Antwort darauf gibt die EULERsche Totientenfunktion $\Phi(m)$ mit:

$$\Phi(26) = 12.$$

Dies sind die 12 Zahlen

$$1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.$$

Daher gibt es $12 \times 26 = 312$ verschiedene Schlüssel für die affine Chiffre.

❸ Die affine Chiffre ist wie die Shift Chiffre eine monoalphabetische Chiffre, da durch die affine Transformation das Klartextaplphabet auf genau ein Chiffre Alphabet abgebildet wird.

❹ Im Beispiel [13.17] ist gezeigt, dass die Affine Chiffre eine Gruppenstruktur besitzt. Dies impliziert, dass eine Hintereinanderausführung zweier Verschlüsselungen mit der Affinen Chiffre und zwei verschiedenen Schlüsseln nicht zu einer Erhöhung der Sicherheit führt.

❺ Das folgende Beispiel [2.3] zeigt die Notwendigkeit der Bedingung $\text{ggT}(\alpha, 26) = 1$ zur Konstruktion der inversen Abbildung (Dechiffrierabbildung).

Beispiel [2.3] Wir wählen als Schlüssel das Zahlenpaar $(\alpha, \beta) = (13, 5)$. Damit lautet die Chiffre:

$$x \longrightarrow (\alpha \cdot x + \beta) \bmod 26 = (13 \cdot x + 5) \bmod 26$$

Der Klartext `hilfe` wird durch diese affine Transformation abgebildet auf:

`h`	$\longrightarrow$	7	$\longrightarrow$	$(13 \cdot 7 + 5) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	`S`
`i`	$\longrightarrow$	8	$\longrightarrow$	$(13 \cdot 8 + 5) \bmod 26 = 5 \bmod 26$	$\longrightarrow$	`F`
`l`	$\longrightarrow$	11	$\longrightarrow$	$(13 \cdot 11 + 5) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	`S`
`f`	$\longrightarrow$	5	$\longrightarrow$	$(13 \cdot 5 + 5) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	`S`
`e`	$\longrightarrow$	4	$\longrightarrow$	$(13 \cdot 4 + 5) \bmod 26 = 5 \bmod 26$	$\longrightarrow$	`F`

Verschlüsselt man den Klartext `rette` mit dieser Chiffre, erhält man den Chiffretext:

`r`	$\longrightarrow$	17	$\longrightarrow$	$(13 \cdot 17 + 5) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	`S`
`e`	$\longrightarrow$	4	$\longrightarrow$	$(13 \cdot 4 + 5) \bmod 26 = 5 \bmod 26$	$\longrightarrow$	`F`
`t`	$\longrightarrow$	19	$\longrightarrow$	$(13 \cdot 19 + 5) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	`S`
`t`	$\longrightarrow$	19	$\longrightarrow$	$(13 \cdot 19 + 5) \bmod 26 = 18 \bmod 26$	$\longrightarrow$	`S`
`e`	$\longrightarrow$	4	$\longrightarrow$	$(13 \cdot 4 + 5) \bmod 26 = 5 \bmod 26$	$\longrightarrow$	`F`

Das heißt also, zwei verschiedene Klartextworte werden durch diese Transformation auf das gleiche Chiffrewort abgebildet. Daher ist eine Umkehrtransformation nicht möglich, da die Abbildung

$$x \longrightarrow (\alpha \cdot x + \beta) \bmod 26 = (13 \cdot x + 5) \bmod 26$$

nicht injektiv ist. Mathematisch liegt der Grund für die nicht Umkehrbarkeit der Abbildung darin, daß $\text{ggT}(a, b) \neq 1$ impliziert, es existiert kein a^{-1} mit $a \cdot a^{-1} \equiv 1 \bmod b$.

Beispiel [2.4]

Mit der affinen Chiffre kann man die Funktionsweise einer **Known–plaintext Attacke** sehr gut demonstrieren. Bei solch einer Angriffsart ist dem Kryptoanalytiker der Chiffretext und der passende Klartext bekannt. Er kann aus diesen Informationen den Schlüssel rekonstruieren.

Angenommen der Angreifer Eve kennt den Klartext `home` und den passenden Chiffretext, der `FCOK` lautet. Eve weiß, dass zur Verschlüsselung die affine Chiffre

$$y \longleftarrow (\alpha \cdot x + \beta) \bmod 26$$

verwendet wurde. Ihr Ziel ist, aus dem bekannten Klartext/Chiffretextpaar den Schlüssel (α, β) zu gewinnen. Sie geht dazu folgendermaßen vor:

① Zunächst werden Klartext und Chiffretext in der üblichen Weise als Zahlen codiert:

$$\texttt{h o m e} \longrightarrow 7\ 14\ 12\ 4$$

und

$$\texttt{F C O K} \longrightarrow 5\ 2\ 14\ 10$$

② Da die affine Chiffre zur Verschlüsselung eingesetzt wurde, bestehen zwischen den Klartext- und Chiffretextzeichen folgende Beziehungen:

$$\begin{aligned} 5 &= (\alpha \cdot 7 + \beta) \bmod 26, \\ 2 &= (\alpha \cdot 14 + \beta) \bmod 26, \\ 14 &= (\alpha \cdot 12 + \beta) \bmod 26, \\ 10 &= (\alpha \cdot 4 + \beta) \bmod 26. \end{aligned}$$

③ Aus zwei beliebigen modularen Gleichungen können nun die Werte α und β berechnet werden.[2]

Multipliziert man die erste modulare Gleichung mit 2 und subtrahiert davon die zweite:

$$\begin{aligned} 10 &= (\alpha \cdot 14 + 2\beta) \bmod 26, \\ -2 &= (-\alpha \cdot 14 - \beta) \bmod 26, \end{aligned}$$

erhält man

$$8 = \beta \bmod 26.$$

[2] Die hier verwendeten Rechenregeln der modularen Arithmetik werden in Kapitel [15.3.1] gerechtfertigt.

Damit ist der erste Parameter gefunden: $\beta = 8$.

Subtrahiert man die zweite Gleichung von der ersten erhält man

$$3 = -7 \cdot a \bmod 26 \equiv 19 \cdot a \bmod 26,$$

oder:

$$19^{-1} \cdot 3 = \alpha \bmod 26.$$

Die Division ist gerechtfertigt, da $\mathrm{ggT}(19, 26) = 1$.
Zu bestimmen ist nun das **multiplikative Inverse** zu 19 modulo 26, i.e. gesucht ist die Zahl $x \in \mathbb{Z}_{26}$ mit

$$19 \cdot x \equiv 1 \bmod 26$$

④ Dazu verwendet Eve den *erweiterten Euklidischen Algorithmus* (siehe dazu Kapitel [18.2]) und erhält

$$19^{-1} \bmod \ = 11,$$

denn

$$11 \cdot 19 = 209 = 8 \cdot 26 + 1.$$

Check:

Initialisierung:

$$\begin{aligned} X_1 &= 1 & Y_1 &= 0 \\ X_2 &= 0 & Y_2 &= 1 \\ X_3 &= 26 & Y_3 &= 19 \end{aligned}$$

Dann ist

$$Q = \left\lfloor \frac{X_3}{Y_3} \right\rfloor = \left\lfloor \frac{26}{19} \right\rfloor = 1.$$

Damit:

$$\begin{aligned} T_1 &= X_1 - Q \cdot Y_1 = 1 \\ T_2 &= X_2 - Q \cdot Y_2 = -1 \\ T_3 &= X_3 - Q \cdot Y_3 = 7 \end{aligned}$$

Ersetzen:

$$\begin{aligned} X_1 &= 0 & Y_1 &= 1 \\ X_2 &= 1 & Y_2 &= -1 \\ X_3 &= 19 & Y_3 &= 7 \end{aligned}$$

Dann ist

$$Q = \left\lfloor \frac{X_3}{Y_3} \right\rfloor = \left\lfloor \frac{19}{7} \right\rfloor = 2.$$

Damit:

$$\begin{aligned} T_1 &= X_1 - Q \cdot Y_1 = -2 \\ T_2 &= X_2 - Q \cdot Y_2 = 3 \\ T_3 &= X_3 - Q \cdot Y_3 = 5 \end{aligned}$$

Ersetzen:

$$\begin{aligned} X_1 &= 1 & Y_1 &= -2 \\ X_2 &= -1 & Y_2 &= 3 \\ X_3 &= 7 & Y_3 &= 5 \end{aligned}$$

Dann ist

$$Q = \left\lfloor \frac{X_3}{Y_3} \right\rfloor = \left\lfloor \frac{7}{5} \right\rfloor = 1,$$

und:

$$\begin{aligned} T_1 &= X_1 - Q \cdot Y_1 = 3 \\ T_2 &= X_2 - Q \cdot Y_2 = -4 \\ T_3 &= X_3 - Q \cdot Y_3 = 2 \end{aligned}$$

Ersetzen:

$$\begin{aligned} X_1 &= -2 & Y_1 &= 3 \\ X_2 &= 3 & Y_2 &= -4 \\ X_3 &= 5 & Y_3 &= 2. \end{aligned}$$

Dann ist

$$Q = \left\lfloor \frac{X_3}{Y_3} \right\rfloor = \left\lfloor \frac{5}{2} \right\rfloor = 2,$$

und:

$$\begin{aligned} T_1 &= X_1 - Q \cdot Y_1 = -8 \\ T_2 &= X_2 - Q \cdot Y_2 = 11 \\ T_3 &= X_3 - Q \cdot Y_3 = 1 \end{aligned}$$

Ersetzen:

$$\begin{aligned} X_1 &= 3 & Y_1 &= -8 \\ X_2 &= -4 & Y_2 &= 11 \\ X_3 &= 2 & Y_3 &= 1 \end{aligned}$$

STOP, der Algorithmus terminiert und $Y_2 = 19^{-1} \bmod 26 = 11$.

⑤ Damit hat Eve den Schlüssel gefunden, er lautet: $(7, 8)$ und die verwendete affine Chiffre ist:

$$y \longleftarrow (7 \cdot x + 8) \bmod 26.$$

2.4 Die Vigenère Chiffre

Eine der wesentlichen Eigenschaften der Shift und der affinen Chiffre ist, dass durch die Wahl eines Schlüssels jedes Klartextzeichen auf genau ein Chiffretextzeichen abgebildet wird. Anders formuliert, das Klartextalphabet wird auf genau ein Chiffretextalphabet abgebildet. Aus diesem Grund nennt man diese Chiffren **monoalphabetische Chiffren**.

In diesem Abschnitt sehen wir uns eine Chiffre an, die nicht monoalphabetisch ist, die sogenannte **Vigenère Chiffre**. Diese Chiffre ist benannt nach Blaise de Vigenère, der im 16. Jahrundert lebte und die Chiffre im Jahre 1586 veröffentlichte. Die Chiffre galt lange als unbrechbar, bis es im 19. Jahrhundert dem englischen Mathematiker Charles Babbage (1791 – 1871) und — unabhängig von Babbage — dem preussischen Offizier Friedrich W. Kasiski (1805 – 1881) gelang,[3] ein Verfahren zu entwickeln, die Vigenère Chiffre zu brechen (siehe [118, pp. 207-213] oder [23]).

Definition [2.5]:

Sei m eine positive ganze Zahl und sei $\{M\} = \{C\} = \{K\} = (\mathbb{Z}_{26})^m$.
Für eine festen Wert (Schlüssel)

$$K = (k_1, k_2, \dots, k_m) = \vec{k}$$

ist die Verschlüsselung die Abbildung

$$\begin{aligned} V_K : (\mathbb{Z}_{26})^m &\longrightarrow (\mathbb{Z}_{26})^m \\ \vec{x} &\longmapsto V_K(\vec{x}) = (\vec{x} + \vec{k}) \bmod 26 \\ &\qquad = \big((x_1 + k_1) \bmod 26, \dots, (x_m + k_m) \bmod 26\big) \end{aligned}$$

Die Dechiffrierung ist die Abbildung

$$\begin{aligned} (V_K)^{-1} : (\mathbb{Z}_{26})^m &\longrightarrow (\mathbb{Z}_{26})^m \\ \vec{y} &\longmapsto (V_K)^{-1}(\vec{x}) = (\vec{x} - \vec{k}) \bmod 26 \\ &\qquad = \big((x_1 - k_1) \bmod 26, \dots, (x_m - k_m) \bmod 26\big) \end{aligned}$$

Benutzt man die übliche Codierung `a` $\longleftrightarrow$ `0`, `b` $\longleftrightarrow$ `1`, usw., dann kann dem Schlüssel K ein aus m Zeichen bestehendes **Schlüsselwort** zugeordnet werden. Die Vigenère Chiffre verschlüsselt immer Klartextstrings der Länge von m Zeichen.

[3] Kasiski entwickelte 1863 einen Test zur Kryptoanalyse der Vigenère Chiffre, der als Kasiski Test bekannt ist. Das von Kasiski entwickelte Verfahren kann die Länge des Schlüsselwortes ermitteln. Dazu wird der Chiffretext nach sich wiederholenden Zeichenfolgen durchsucht. Aus den Abständen zwischen diesen Zeichenfolgen wird der größte gemeinsame Teiler berechnet, der dann die vermutliche Schlüssellänge ergibt.

Beispiel [2.5]

Die beiden Protagonisten Alice und Bob wollen eine mit der VIGENÈRE Chiffre verschlüsselte Nachricht austauschen. Da die VIGENÈRE Chiffre ebenfalls ein symmetrisches Verfahren ist, müssen Alice und Bob zuvor auf sicherem Weg den Schlüssel austauschen. Das bedeutet in diesem Fall, sie müssen ein nur ihnen bekanntes Schlüsselwort wählen, zum Beispiel

`jamesbond`.

Da dieses Schlüsselwort aus neun Zeichen besteht, ist $m = 9$. Die Schlüsselwortzeichen werden gemäß dem Standardverfahren in Zahlen codiert, damit erhält man:

`jamesbond` $\longrightarrow$ `9 0 12 4 18 1 14 13 3`.

Mit Hilfe dieses Schlüssels wird nun der Klartext wie folgt verschlüsselt. Der Klartext, den Alice an Bob senden will, ist

`treffenummitternacht`.

Alice codiert den Klartext zunächst in Zahlenform und schreibt den Schlüssel unter die sich ergebenden Zahlen:

t	r	e	f	f	e	n	u	m	m	i	t	t	e	r	n	a	c	h	t
19	17	4	5	5	4	13	20	12	12	8	19	19	4	17	13	0	2	7	19
9	0	12	4	18	1	14	13	3	9	0	12	4	18	1	14	13	3	9	0

Die Zahlenspalten werden nun modulo 26 addiert. Die resultierenden Zahlen entsprechen dann dem Chiffretext.

t	r	e	f	f	e	n	u	m	m	i	t	t	e	r	n	a	c	h	t
19	17	4	5	5	4	13	20	12	12	8	19	19	4	17	13	0	2	7	19
9	0	12	4	18	1	14	13	3	9	0	12	4	18	1	14	13	3	9	0
2	17	16	9	23	5	1	7	15	21	8	5	23	22	18	1	13	5	16	19
C	R	Q	J	X	F	B	H	P	V	I	F	X	W	S	B	N	F	Q	T

Damit lautet der Chiffretext:

`CRQJXFBHPVIFXWSBNFQT`

Zum Entschlüsseln benutzt Bob das gleiche Schlüsselwort, subtrahiert dieses aber modulo 26 vom Chiffretext.

Anmerkungen:

❶ Die Anzahl der möglichen Schlüsselworte der Länge m einer VIGENÈRE Chiffre — *i.e.* die Größe des Schlüsselraums — ist 26^m. Ein relativ kleines Schlüsselwort z.B. der Länge $m = 5$ impliziert ein Schlüsselraum mit etwa $1.18 \cdot 10^7$ Elementen. Dies ist bereits ausreichend, um eine manuelle Brute-Force Attacke auszuschließen. Mit Computern ist dies jedoch kein Problem.

❷ In einer VIGENÈRE Chiffre mit Schlüsselwort der Länge m kann ein Klartextzeichen auf eines von m möglichen Chiffretextzeichen abgebildet werden. Dabei ist angenommen, dass das Schlüsselwort auch m verschiedene Zeichen enthält. Damit ist die VIGENÈRE Chiffre eine typische **polyalphabetische Chiffre**.

Alternativ zur Addition modulo 26 kann die VIGENÈRE Verschlüsselung auch mit Hilfe einer Tabelle durchgeführt werden. Dazu betrachten wir die in der Abbildung [2.1] aufgelisteten Alphabete. Ist der zu verschlüsselnde Klartext `treffenum..` und das Passwort `jamesbond`, dann kann die Verschlüsselung folgendermaßen durchgeführt werden:

- Klartextzeichen `t` ⟶ Spalte `t`; Passwortzeichen `j` ⟶ Zeile `j`.
  Der Schnittpunkt Spalte `t` und Zeile `j` hat das Zeichen `C` ⟶ Chiffrezeichen `C`.
- Klartextzeichen `r` ⟶ Spalte `r`; Passwortzeichen `a` ⟶ Zeile `a`.
  Der Schnittpunkt Spalte `r` und Zeile `a` hat das Zeichen `R` ⟶ Chiffrezeichen `R`.
- Klartextzeichen `e` ⟶ Spalte `e`; Passwortzeichen `m` ⟶ Zeile `m`.
  Der Schnittpunkt Spalte `e` und Zeile `m` hat das Zeichen `Q` ⟶ Chiffrezeichen `Q`.
- usw.

Die Dechiffrierung mit Hilfe der VIGENÈRE Tabelle funktioniert folgendermaßen. Angenommen, das Passwort ist wieder `jamesbond` und der empfangene Chiffretext lautet `CRQJXFBHPVIFXWSBNFQT`.

- Passwortzeichen `j` ⟶ Zeile `j`. Chiffrezeichen `C` ⟶ gehe in der Zeile `j` solange nach rechts, bis das Zeichen `c` auftritt. Die zugehörige Spaltenüberschrift ist das gesuchte Klartextzeichen: `t`.
- Passwortzeichen `a` ⟶ Zeile `a`. Chiffrezeichen `R` ⟶ gehe in der Zeile `a` solange nach rechts, bis das Zeichen `r` auftritt. Die zugehörige Spaltenüberschrift ist das gesuchte Klartextzeichen: `r`.
- Passwortzeichen `m` ⟶ Zeile `m`. Chiffrezeichen `Q` ⟶ gehe in der Zeile `m` solange nach rechts, bis das Zeichen `q` auftritt. Die zugehörige Spaltenüberschrift ist das gesuchte Klartextzeichen: `e`.
- usw.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
b	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
c	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
d	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
e	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
f	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
g	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
h	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
i	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
j	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
k	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
l	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
m	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
n	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
o	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
p	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
r	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
s	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
t	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
u	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
v	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
w	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
x	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Tabelle 2.1: Die VIGENÈRE Tabelle.

2.5 Die Playfair Chiffre

Die PLAYFAIR Chiffre ist eine Blockchiffre, die Digramme des Klartextes in einem Vorgang in Digramme des Chiffretexts abbildet. Die PLAYFAIR Chiffre wurde 1854 von SIR CHARLES WHEATSTONE entwickelt, ist aber nach dessen Freund Baron PLAYFAIR OF ST. ANDREWS benannt; siehe [206, pp. 41], [215, Chap. 2.6] oder [11] pp. 168. Die PLAYFAIR Chiffre wurde von den Briten im Burenkrieg und während des Ersten Weltkriegs benutzt. Die Australier benutzten diese Chiffre aufgrund ihrer Einfachheit — es ist keine spezielle Ausrüstung nötig — für taktische Zwecke währen des Zweiten Weltkriegs.

Der PLAYFAIR Algorithmus basiert auf einer 5×5 Matrix, die aus Buchstaben

Abbildung 2.2: SIR CHARLES WHEATSTONE (links) und LORD PLAYFAIR.

konstruiert wird. Der Schlüssel der PLAYFAIR Chiffre bildet ein *Schlüsselwort*. Als Beispiel benutzen wir das Schlüsselwort *monarchy*.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Die Matrix wird folgendermaßen konstruiert: Die Buchstaben des Schlüsselworts (abzüglich Duplikate) werden von links nach rechts und von oben nach unten in die Matrix eingetragen. Der Rest der Matrix wird mit den verbleibenden Buchstaben in alphabetischer Ordnung gefüllt. Die Buchstaben i und j zählen als ein Buchstabe. Bei der Verschlüsselung des Klartexts werden zwei Klartextzeichen in einem Schritt chiffriert. Dies geschieht nach folgenden Regeln:

① Sich wiederholende Klartextzeichen in einem Paar – oder als Füller um auf eine gerade Anzahl von Klartextzeichen zu kommen – werden mit einem Füllzeichen – z.B. x – getrennt. Dadurch wird aus `mannheim` das zu chiffrierende Wort `ma nx nh ei mx`.

② Zwei Klartextzeichen, die in die gleiche Zeile der Matrix liegen werden jeweils durch das rechts davon liegende Zeichen ersetzt. Wenn eines der

Klartextzeichen dabei in der Spalte ganz rechts auftritt, wird es durch das erste Zeichen der Zeile ersetzt. Im obigen Beispiel wird das Digram `ar` daher durch `RA` ersetzt.

③ Zwei Klartextzeichen, die in die gleiche Spalte der Matrix liegen werden jeweils durch das darunter liegende Zeichen ersetzt. Wenn eines der Zeichen in der letzten Zeile auftritt, wird es durch das erste Zeichen der Spalte ersetzt. Im obigen Beispiel wird daher das Digramm `mu` durch den Block `CM` substituiert.

④ In allen anderen Fällen wird jeder Klartextbuchstaben eines Paars ersetzt durch das Zeichen, das in der gleichen Zeile und der Spalte liegt, in der das andere Klartextzeichen liegt. Im obigen Beispiel wird daher das Klartextdigram `hs` durch `BP` und `ea` durch `IM` ersetzt.

Beispiel:

In einem Beispiel verwenden wir das Schlüsselwort `logarithm`. Dies führt auf die Matrix:

L	O	G	A	R
I	T	H	M	B
C	D	E	F	K
N	P	Q	S	U
V	W	X	Y	Z

Der zu chiffrierende Klartext ist

```
come quickly we need help
```

Damit sind also folgende Digramme gemäß den obigen Regeln zu chiffrieren:

```
co me qu ic kl yw en ex ed he lp
```

Damit haben wir:

Klartext-digramm	**Regel**	**Chiffre-Digramm**
co	④	DL
me	④	FH
qu	②	SN
ic	③	CN
kl	④	CR
yw	②	ZX
en	④	CQ
ex	③	QG
ed	②	FE
he	③	EQ
lp	④	ON

Damit lautet der Chiffretext:

```
DLFHSNCNCRZXCQQGFEEQON
```

Die Dechiffrierung wird genau in umgekehrter Reihenfolge ausgeführt, i.e. die folgenden Regeln sind hier anzuwenden:

① Zwei Klartextzeichen, die in die gleiche Zeile der Matrix liegen werden jeweils durch das *links* davon liegende Zeichen ersetzt. Wenn eines der Klartextzeichen dabei in der Spalte ganz links auftritt, wird es durch das letzte Zeichen der Zeile ersetzt.

② Zwei Klartextzeichen, die in die gleiche Spalte der Matrix liegen werden jeweils durch das darüber liegende Zeichen ersetzt. Wenn eines der Zeichen in der ersten Zeile auftritt, wird es durch das letzte Zeichen der Spalte ersetzt.

③ In allen anderen Fällen wird jeder Klartextbuchstaben eines Paars ersetzt durch das Zeichen, das in der gleichen Spalte und der Zeile liegt, in der das andere Klartextzeichen liegt.

Für den Chiffretext

```
DL FH SN CN CR ZX CQ QG FE EQ ON
```

führt dies mit der Matrix

L	O	G	A	R
I	T	H	M	B
C	D	E	F	K
N	P	Q	S	U
V	W	X	Y	Z

auf folgendes Verfahren:

Chiffre-digramm	Regel	Klartext-Digramm
DL	③	co
FH	③	me
SN	①	qu
CN	②	ic
CR	③	kl
ZX	①	yw
CQ	③	en
QG	②	ex
FE	①	ed
EQ	②	he
ON	③	lp

Damit haben wir den klaretext

```
come quickly we need help
```

rekonstruiert.

Anmerkungen:

1. Die PLAYFAIR Chiffre ist eine Substitutionschiffre.
2. Der Chiffretext enthält immer eine gerade Anzahl von Zeichen.
3. Kein Klartextbuchstabe wird mit der PLAYFAIR Chiffre auf sich selbst abgebildet.

Eine Möglichkeit, die Effektivität der PLAYFAIR Chiffre – und anderer Chiffren – zu untersuchen, ist in der Abbildung [2.3] dargestellt. Die durchgezogene Linie, die mit `plaintext` beschriftet ist, ist ein Plot der Häufigkeit der 26 Buchstaben in dem *Encyclopaedia Britannica* Artikel über Kryptologie. Dies ist gleichzeitig die Häufigkeitsverteilung irgendeiner monoalphabetischen Substitutionschiffre. Der Plot aus Abbildung [2.3] entsteht folgendermaßen: Man zählt die Anzahl des Auftretens jedes Buchstabens im besagten Artikel und dividiert anschließend diese Anzahl durch den Wert des Buchstabens e, das ist der häufigste Buchstabe. Als Folge dessen hat e die relative Häufigkeit 1, der Buchstabe t etwa 0.76, usw. Die Punkte der horizontalen Achse entsprechen den Buchstaben des Alphabets in der Reihenfolge der abnehmenden Häufigkeit.

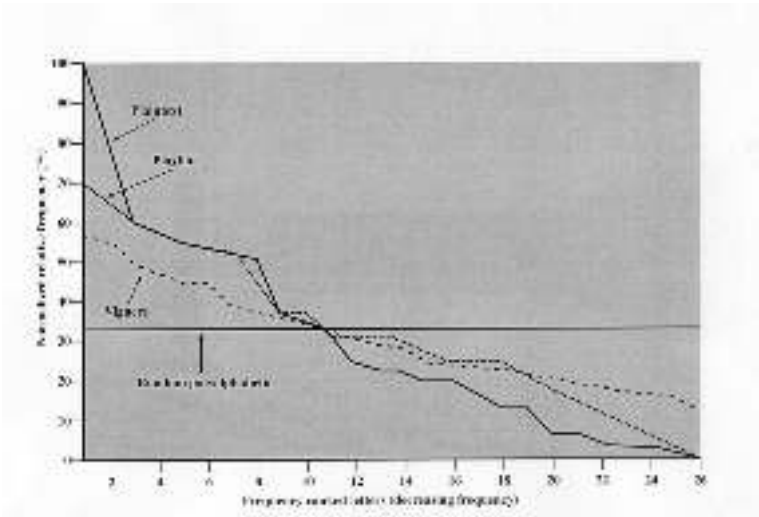


Abbildung 2.3: Relative Häufigkeit des Auftretens von Buchstaben [206, p. 42].

Die Abbildung [2.3] zeigt auch die Häufigkeitsverteilung der Buchstaben, wenn Klartext mit der PLAYFAIR Chiffre verschlüsselt wird. Um den Plot zu normalisieren, wurde die Anzahl des Auftretens jedes Buchstabens wieder mit der Häufigkeit des Buchstabens e im Klartext dividiert. Der resultierende Plot (i.e. die gepunktete Linie) zeigt daher das Ausmaß dessen, wie gut die Häufigkeitsverteilung der Buchstaben durch den Verschlüsselungsprozess maskiert wird.

Wenn die Information der Häufigkeitsverteilung der Buchstaben durch den Verschlüsselungsvorgang komplett verborgen wird, dann muß die Chiffre als einen gleichverteilten Output produzieren. Das ist in der Abbildung [2.3] die waagrechte Linie, die mit `random polyalphabetic cipher` beschriftet ist. In diesem Fall ist eine Kryptoanalyse eines Chiffretextes – wobei nur der Chiffretext bekannt ist – effektiv nicht möglich. Wie die Abbildung zeigt, hat die PLAYFAIR Chiffre eine flachere Kurve als der Klartext, dennoch zeigt diese Chiffre genug Struktur für die Kryptoanalyse.

2.6 Die Hill Chiffre

In diesem Abschnitt sehen wir uns eine weitere polyalphabetische Chiffre an, die im Jahre 1929 von LESTER S. HILL eingeführt wurde.[4] Diese Chiffre war nie sehr verbreitet in Gebrauch, ihre Bedeutung liegt vielmehr darin, dass erstmals von algebraischen Methoden[5] Gebrauch gemacht wurde.

Die Verschlüsselung mit der HILL Chiffre besteht aus folgenden Einzelschritten, die wir hier anhand eines konkreten Beispiels durchexerzieren:

① *Auswahl des Schlüssels*
Man wählt eine Zahl, zum Beispiel $n = 3$. Der Schlüssel des Verfahrens ist eine $n \times n$ Matrix M. Die Einträge der Matrix sind Zahlen modulo 26. Als Beispiel wählen wir die Matrix

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{pmatrix}$$

② *Verschlüsselung*
Der zu verschlüsselnde Klartext wird nun in eine Anzahl von Zeilenvektoren aufgespaltet. Die Länge dieser Vektoren ist n, *i.e.* der Klartext wird in **Blöcke** von n Zeichen aufgeteilt, in unserem Beispiel $n = 3$. Der zu verschlüsselnde Klartext ist zum Beispiel:

```
berlin
```

Die Standardcodierung liefert die Ziffernfolge

```
1 4 17 11 8 13
```

Diese Ziffernfolge wird nun in zwei Blöcke mit je drei Ziffern aufgespaltet. Der erste Block ist:

$$\begin{pmatrix} 1 & 4 & 17 \end{pmatrix}$$

Dieser Zeilenvektor wird nun mit der Matrix M (von rechts) multipliziert (modulo 26), so daß ein neuer Zeilenvektor entsteht.

$$\begin{pmatrix} 1 & 4 & 17 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 204 & 175 & 163 \end{pmatrix} \bmod 26 \equiv \begin{pmatrix} 22 & 19 & 7 \end{pmatrix}$$

Der zweite Block liefert in analoger Weise:

$$\begin{pmatrix} 11 & 8 & 13 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{pmatrix} \equiv \begin{pmatrix} 4 & 23 & 3 \end{pmatrix}$$

Damit erhalten wir durch Aneinanderfügen der Blöcke $(22, 19, 7)$ und $(4, 23, 3)$ den Chiffretext:

[4] Siehe die Monographien [213, 215], [11], pp 227 und HILL's Orginalarbeit [108].
[5] Insbesondere Lineare Algebra und modulare Arithmetik.

$$\texttt{berlin} \longrightarrow (22, 19, 7, 4, 23, 3) \longrightarrow \texttt{WTHEXD}$$

③ *Dechiffrierung*
Zur Entschlüsselung eines mit der HILL Chiffre verschlüsselten Textes benötigt man die zu M inverse Matrix, die wir N nennen. Diese Matrix erfüllt:

$$M \cdot N \equiv 1_{3\times 3} \bmod 26$$

Für die Existenz der Umkehrmatrix N muß die Determinate der Matrix M die Bedingung erfüllen:

$$\texttt{ggT}\big(\texttt{det}(M), 26\big) = 1$$

Nun ist:

$$\begin{aligned}\texttt{det}(M) &= \texttt{det}\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 11 & 9 & 8\end{pmatrix}\\ &= 40 + 132 + 108 - 165 - 64 - 54\\ &= -3\end{aligned}$$

Daher

$$\texttt{det}(M) = -3$$

$$M^{-1} = \frac{1}{-3}\begin{pmatrix}-14 & 11 & -3\\ 34 & -25 & 6\\ -19 & 13 & -3\end{pmatrix}$$

Da

$$-3 \equiv 23 \bmod 26$$

ist das multiplikative Inverse zu 23 modulo 26 zu bestimmen. Mit Hilfe des erweiterten EUKLIDischen Algorithmus erhält man:

$$23 \cdot 17 \equiv 1 \bmod 26$$

Daher ersetzen wir den Bruch (-1/3) durch 17, multiplizieren die Matrixelemente mit 17 und reduzieren mod 26. Damit:

$$N = M^{-1} = \begin{pmatrix}22 & 5 & 1\\ 6 & 17 & 24\\ 15 & 13 & 1\end{pmatrix}$$

Wir verifizieren, daß gilt:

$$M \cdot N \equiv 1_{3\times 3} \bmod 26$$

$$
\begin{aligned}
M \cdot N &= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{pmatrix} \cdot \begin{pmatrix} 22 & 5 & 1 \\ 6 & 17 & 24 \\ 15 & 13 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 22+12+45 & 5+34+39 & 1+48+3 \\ 88+30+90 & 20+85+78 & 4+120+6 \\ 242+54+120 & 55+153+104 & 11+216+8 \end{pmatrix} \\
&= \begin{pmatrix} 79 & 78 & 52 \\ 208 & 183 & 130 \\ 416 & 312 & 235 \end{pmatrix} \mod 26 \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= 1_{3\times 3}
\end{aligned}
$$

Der Empfänger des Chiffretexts codiert den erhaltenen Textstring gemäß dem Standardverfahren in Zahlen zwischen 0 und 26 und teilt anschließend die Zahlenfolge in 3er Blöcke. Die Dechiffrierung erfolgt wieder durch Matrizenmultiplikation.

$$\texttt{WTHEXD} \longrightarrow (22, 19, 7, 4, 23, 3)$$

Es folgt:

$$
\begin{aligned}
\begin{pmatrix} 22 & 19 & 7 \end{pmatrix} \cdot N &= \begin{pmatrix} 22 & 19 & 7 \end{pmatrix} \cdot \begin{pmatrix} 22 & 5 & 1 \\ 6 & 17 & 24 \\ 15 & 13 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 484+114+105 & 110+323+91 & 22+456+7 \end{pmatrix} \mod 26 \\
&= \begin{pmatrix} 703 & 524 & 485 \end{pmatrix} \mod 26 \\
&= \begin{pmatrix} 1 & 4 & 17 \end{pmatrix} \mod 26
\end{aligned}
$$

Für den zweiten 3er Block folgt:

$$
\begin{aligned}
\begin{pmatrix} 4 & 23 & 3 \end{pmatrix} \cdot N &= \begin{pmatrix} 4 & 23 & 3 \end{pmatrix} \cdot \begin{pmatrix} 22 & 5 & 1 \\ 6 & 17 & 24 \\ 15 & 13 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 88+138+45 & 20+391+39 & 4+552+3 \end{pmatrix} \mod 26 \\
&= \begin{pmatrix} 271 & 450 & 559 \end{pmatrix} \mod 26 \\
&= \begin{pmatrix} 11 & 8 & 13 \end{pmatrix} \mod 26
\end{aligned}
$$

Schließlich werden die 3er Blöcke konkateniert und decodiert:

$$(1, 4, 17, 11, 8, 13) \longrightarrow \texttt{berlin}$$

Da die HILL Chiffre ganze Klartextblöcke in einem Vorgang verschlüsselt, ist diese Verschlüsselung ein Beispiel einer sogenannten Block Chiffre.

Definition [2.6]:

Sei $n \geq 2$ und sei $\{M\} = \{C\} = (\mathbb{Z}_{26})^n$. Weiterhin sei

$$\{K\} = \{\text{Invertierbare } n \times n \text{ Matrizen über } \mathbb{Z}_{26}\}$$

Die Verschlüsselung ist die Matrizenmultiplikation mit $M \in \{K\}$ mit

$$e_M(x) = y = x \cdot M$$

Die Dechiffrierung ist die Abbildung

$$d_M(y) = y \cdot M^{-1}$$

wobei alle Operationen modulo 26 auszuführen sind.

Anmerkung:

Die HILL Chiffre hat die Eigenschaft der **Diffusion**, *i.e.* die Änderung eine Klartextzeichens hat zur Folge, dass mehrere Chiffretextzeichen geändert werden. Dies demonstriert das folgende Beispiel. Wir verschlüsseln mit obiger Matrix M die beiden Klartextblöcke `aab` und `abb`, die sich in einem Zeichen unterscheiden. Man erhält:

$$\begin{aligned}
\texttt{aab} &\longrightarrow (0\ 0\ 1) \\
&\longrightarrow (0\ 0\ 1) \cdot M \\
&= (0\ 0\ 1) \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{pmatrix} \\
&= (11\ 9\ 8) \\
&\longrightarrow \texttt{LJI}.
\end{aligned}$$

Andererseits ergibt der Klartextblock `abb` den folgenden Chiffreblock:

$$\begin{aligned}
\texttt{abb} &\longrightarrow (0\ 1\ 1) \\
&\longrightarrow (0\ 1\ 1) \cdot M \\
&= (0\ 1\ 1) \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{pmatrix} \\
&= (15\ 14\ 14) \\
&\longrightarrow \texttt{POO}.
\end{aligned}$$

2.7 Die ADFGVX Chiffre

Der deutsche Nachrichtenoffizier Oberst FRITZ NEBEL entwickelte während des ersten Weltkriegs ein Verfahren, das am 1. März 1918 erstmals eingesetzt wurde, die ADFGX-Verschlüsselung. In der Folge wurde die Chiffre weiterentwickelt und es entstand die **ADFGVX**–Chiffre, die ab Juni 1918 von den Deutschen eingesetzt wurde.[6]

Die ADFGX- und die ADFGVX-Chiffre sind zweistufig aufgebaut und verwenden zum Chiffrieren in einem ersten Schritt eine monoalphabetische Substitutionsoperation und anschließend eine Vertauschung der Buchstaben, also eine Permutationsoperation.

Chiffrierung mit der ADFGVX–Chiffre

1. Schritt: Substitution

Für den ersten Teil der Verschlüsselung, die Substitution wird ein sogenanntes **Polybios–Quadrat** erstellt.[7] Dazu werden die Buchstaben **A, D, F, G , V, X** in die Spalten und Zeilenköpfe gesetzt. Der Grund dafür, dass für diese Chiffre gerade diese Buchstaben gewählt wurden ist, dass sich die Zeichen `A, D, F, G, V` und `X` als Codeworte im MORSE–Code deutlich voneinander unterscheiden und damit die Gefahr von Übertragungsfehlern bei der Übertragung in Funksprüchen deutlich verringert.

	A	**D**	**F**	**G**	**V**	**X**
A	h	i	n	t	e	r
D	a	l	b	c	d	f
F	g	j	k	m	o	p
G	q	s	u	v	w	x
V	y	z	0	1	2	3
X	4	5	6	7	8	9

Dadurch entsteht ein Quadrat mit 36 Einträgen. In diese Einträge werden nun die 26 Buchstaben des lateinischen Alphabets und die 10 Ziffern 0 bis 9 eingetragen. Die Art und Weise, wie die Zeichen angeordnet werden, ist Teil des Schlüssels der ADFGVX-Verschlüsselung. Eine Möglichkeit besteht darin, ein Schlüsselwort zu wählen, beispielsweise `hinterhalt`. Man beginnt mit dem Füllen des Quadrats mit dem Schlüsselwort, wobei mehrfach auftretende Buchstaben nur einmal verwendet werden. Daher wird in das Quadrat `hinteral` eingetragen. Der Rest der freien Einträge wird mit den verbliebenen Buchstaben und den Ziffern gefüllt.

[6] Siehe zum Beispiel [204, Anhang F], [11, pp. 185] oder [113, pp.116].

[7] Dieses Quadrat ist benannt nach dem griechischen Geschichtsschreiber POLYBIOS (ca. 200 – 120 v. Chr.), der in seinem überlieferten Werk *Historiai* ein ähnliches Verfahren beschrieb. Bei dieser sogenannten **Polybios–Chiffre** wird ein Quadrat erstellt mit den Ziffern 1 bis 5 als Zeilen– und Spaltenbeschriftung. Anschließend wird das Quadrat mit den Zeichen des Alphabets gefüllt. Das Chiffrezeichen eines Buchstabens ist dann einfach das Zahlenpaar Zeile/Spalte, in der der Buchstabe zu finden ist. Das so entstandene Zahlenpaar wurde in der Antike durch optische Übertragung (Fackeln) übermittelt. Siehe dazu auch die Monographie von CRAIG BAUER [12], p. 98.

Wir wollen mit diesem Verfahren den Klartext

`informatik`

verschlüsseln. Jeder Klartextbuchstabe wird durch ein Buchstabepaar ersetzt (Substitution), bestehend aus Zeilen- und Spaltenbeschrifter. Dadurch wird aus `i` das Paar `AD`, aus `n` wird `AF` usw. Es ergibt sich der folgende Zwischentext:

i	n	f	o	r	m	a	t	i	k
AD	AF	DX	FV	AX	FG	DA	AG	AD	FF

2. Schritt: Transposition

Der zweite Schritt besteht nun aus einer Permutation, dazu wird ein zweites Schlüsselwort benutzt, beispielsweise `jamesbond`.

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3
A	D	A	F	D	X	F	V	A
x	F	G	D	A	A	G	A	D
F	F							

Der Chiffretext ist nun die Aneinanderreihung der Spalten in der Reihenfolge der Nummerierung der Schlüsselwortbuchstaben. Zuerst wird also die Spalte unter dem Buchstaben `a` hingeschrieben, das sind die Zeichen `DFF`. Dann folgt die Spalte unter dem `b`, das sind die Zeichen `XA`, usw. Damit ergibt sich der folgende Chiffretext:

`DFFXAADFDAXFAGVAFGDA`

Dieser Text wird anschließend im MORSE–Code gefunkt.

Dechiffrierung der ADFGVX-Chiffre

Der Empfänger eines mit der ADFGVX-Chiffre chiffrierten Funkspruchs verfügt über die beiden Teilschlüssel,[8] in unserem Beispiel `hinterhalt` und `jamesbond` und natürlich den empfangenen Chiffretext

`DFFXAADFDAXFAGVAFGDA.`

Der Empfänger des Chiffrats muss nun die beiden Schritte des Chiffrieres in umgekehrter Reihenfolge durchführen.

In einem ersten Schritt wird abgezählt, aus wievielen Zeichen der empfangene Chiffretext besteht, in dem konkreten Beispiel sind das 20 Zeichen. Dann werden die Anzahl der Zeichen des zweiten Schlüsselwortes für die Transposition (`jamesbond`) gezählt, hier 9 Zeichen. Bildet der Dechiffrierer nun die ganzzahlige Division

[8] Die Schlüssel wurden täglich geändert.

Anzahl Zeichen Chiffretext / Anzahl Zeichen Schlüsselwort = Anzahl ganze Zeilen + Rest

dann erhält er die Information, wieviele ganze Zeilen und restliche Zeichen unter das Schlüsselwort zu schreiben sind.

Beispiel [2.6]

Das empfangene Chiffrat `DFFXAADFDAXFAGVAFGDA` besteht aus 20 Zeichen, das Schlüsselwort `jamesbond` aus 9 Zeichen. Dann ist

$$20 \text{ div } 9 = 2 \text{ Rest } 2.$$

Der erste Schritt bei der Entschlüsselung besteht darin, das Schlüsselwort

`jamesbond`

aufzuschreiben und die Buchstaben zu nummeriene.

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3

Unter dem Schlüsselwort wird nun das Chiffrat in zwei komplette Zeilen geschrieben, die die ersten beiden (das ist der Rest 2) Spalten aus drei Zeilen bestehen. Also `a` steht in Spalte 2, daher werden die ersten drei Buchstaben des Chiffrats in diese Spalte geschrieben:

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3
	D							
	F							
	F							

Das Zeichen 2, also `b` steht in Spalte 6, daher werden die nächsten beiden Zeichen des Chiffrats unter `b` geschrieben:

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3
	D				X			
	F				A			
	F							

Zeichen 3 ist das `d` in Spalte 9, hier werden die nächsten beiden Zeichen eingetragen:

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3
	D				X			A
	F				A			D
	F							

Zeichen 4 ist `e` in Spalte 4, hier werden wieder 2 Zeichen des Chiffretexts eingetragen:

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3
	D		F		X			A
	F		D		A			D
	F							

Das Zeichen 5 ist `j` in der ersten Spalte. Daher müssen jetzt die folgenden drei Zeichen eingetragen werden.

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3
A	D		F		X			A
X	F		D		A			D
F	F							

Auf diese Weise komplettiert der Empfänger die Tabelle und erhält:

j	a	m	e	s	b	o	n	d
5	1	6	4	9	2	8	7	3
A	D	A	F	D	X	F	V	A
X	F	G	D	A	A	G	A	D
F	F							

Damit erhält der Empfäger also die Buchstabenpaare:

```
AD AF DX FV AX FG DA AG AD FF
```

Im nächsten Schritt erstellt der Empfänger das Polybios-Quadrat aus dem zweiten Schlüsselwort, das er natürlich kennen muß. Mit dem Schlüsselwort

```
hinterhalt
```

ergibt sich dabei wieder die folgende Tabelle:

	A	**D**	**F**	**G**	**V**	**X**
A	h	i	n	t	e	r
D	a	l	b	c	d	f
F	g	j	k	m	o	p
G	q	s	u	v	w	x
V	y	z	0	1	2	3
X	4	5	6	7	8	9

Daraus liest man ab:

AD	i
AF	n
DX	f
FV	0
AX	r
FG	m
DA	a
AG	t
AD	i
FF	k

2.8 Permutations Chiffre

Alle bisher untersuchten Kryptosysteme stellen sogenannte **Substitutionschiffren** dar. Das bedeutet, die Klartextzeichen bleiben an den ursprünglichen Positionen, werden aber durch andere Zeichen ersetzt. Die grundlegende Idee einer **Permutationschiffre** ist, dass die Form der einzelnen Klartextzeichen erhalten bleibt, die Positionen der Zeichen im Klartext aber geändert werden.

Definition [2.7]:

Eine **Permutation** einer endlichen Menge M ist eine bijektive Abbildung

$$\pi \ : \ M \longrightarrow M$$

Mit anderen Worten, eine Permutation ist eine injektive und surjektive Abbildung einer Menge M auf sich selbst. Dies impliziert, dass für jedes Element $x \in M$ ein Element $x' \in M$ existiert, so dass

$$\pi(x') = x$$

Daher können wir die Umkehrabbildung betrachten, dies ist die *inverse Permutation*.

$$\pi^{-1} \ : \ M \longrightarrow M$$

mit:

$$\pi^{-1}(x) = x' \text{ genau dann, wenn } \pi(x') = x.$$

Die **Permutations Chiffre** — auch unter Transpositionschiffre bekannt — wird formal folgendermaßen definiert:

Definition [2.8]:

Sei n eine positive ganze Zahl und sei $\{M\} = \{C\} = (\mathbb{Z}_{26})^n$. Der Schlüsselraum $\{K\}$ besteht aus allen Permutationen der Menge $\{1, 2, \ldots, n\}$. Für einen Schlüssel π — *i.e.* eine bestimmte Permutation — definieren wir die Verschlüsselung durch:

$$e_\pi(x_1, x_2, \ldots, x_n) = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

Die Dechiffrierung erfolgt durch

$$d_\pi(y_1, y_2, \ldots, y_n) = (y_{\pi^{-1}(1)}, y_{\pi^{-1}(2)}, \ldots, y_{\pi^{-1}(n)})$$

wobei π^{-1} die zu π inverse Permutation ist.

Bei der Benutzung der Permutations Chiffre verwendet man üblicherweise nicht die Codierung der Klartext– bzw. Chiffretextzeichen durch Zahlen, da bei der Ver– bzw. Entschlüsselung keine algebraischen Operationen durchgeführt werden.

Beispiel [2.7]

Wir wählen $n = 6$ und als Schlüssel die folgende Permutation:

$$\pi(x) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 1 & 6 & 4 & 2 \end{pmatrix}$$

Die erste Zeile dieser Darstellung der Permutation listet die Wert von x auf mit $1 \leq x \leq 6$, die zweite Zeile die entsprechenden Werte von $\pi(x)$. Die inverse Permutation erhält man durch:

1. Vertauschung der beiden Zeilen
2. Neuordnung der Spalten so, daß die erste Zeile in aufsteigender Ordnung ist

Damit

$$\pi^{-1}(x) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 1 & 5 & 2 & 4 \end{pmatrix}$$

Der mit der Permutations Chiffre zu verschlüsselnde Klartext sei nun

`berufsakademiemannheim`

Dieser Klaretxt wird zunächst in Blöcke der Länge von $n = 6$ Zeichen aufgeteilt:

`berufs` | `akadem` | `iemann` | `heim`

Jede Gruppe der sechs Buchstaben wird nun gemäß dem Schlüssel permutiert. Sehen wir uns im Detail an, wie dies an dem Block `berufs` funktioniert. Nach Wahl des Schlüssels wird das Klartextzeichen der ersten Position auf die dritte Position im Chiffretext gesetzt, das zweite Zeichen auf die fünfte, das dritte auf die erste, das vierte auf die sechste, das fünfte Klartextzeichen auf die vierte Position und schließlich das letzte Zeichen des Blocks auf die zweite Position. Dies ergibt daher den folgenden Chiffretextblock:

$$\texttt{berufs} \xrightarrow{\pi} \texttt{RSBFEU}$$

In der gleichen Weise wird der zweite Block `akadem` abgebildet auf:

$$\texttt{akadem} \xrightarrow{\pi} \texttt{AMAEKD}$$

Der dritte Klartextblock `iemann` wird zu

$$\texttt{iemann} \xrightarrow{\pi} \texttt{MNINEA}$$

und der Rest

$$\texttt{heim} \xrightarrow{\pi} \texttt{IHME}$$

Damit lautet der komplette Chiffretext:

$$\texttt{RSBFEUAMAEKDMNIENAIHME}$$

Die Dechiffrierung erfolgt mit der inversen Permutation. Zunächst wird der Chiffretext in Blöcke zu je sechs Zeichen aufgespaltet:

$$\texttt{RSBFEU} \mid \texttt{AMAEKD} \mid \texttt{MNINEA} \mid \texttt{IHME}$$

Der erste Block wird mit der Permutation

$$\pi^{-1}(x) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 1 & 5 & 2 & 4 \end{pmatrix}$$

auf den Klartext

$$\texttt{RSBFEU} \xrightarrow{\pi^{-1}} \texttt{berufs}$$

abgebildet. Analog erhält man für den zweiten Block:

$$\texttt{AMAEKD} \xrightarrow{\pi^{-1}} \texttt{akadem}$$

den dritten

$$\texttt{MNINEA} \xrightarrow{\pi^{-1}} \texttt{iemann}$$

und schließlich

$$\texttt{IHME} \xrightarrow{\pi^{-1}} \texttt{heim}$$

woraus durch Konkatenation der einzelnen Blöcke der komplette Klartext folgt:

$$\texttt{berufsakademiemannheim}$$

2.9 Produkt Chiffren

Eine naheliegende Methode, bekannte Verschlüsselungsverfahren stärker zu machen, besteht darin, verschiedene Verschlüsselungsverfahren (in geeigneter Weise) zu kombinieren. Ein einfache Variante einer Produkt Chiffre besteht aus

- einem Codebook
- einer Shift-Chiffre

Ein Codebuch ist ein Dokument, in dem Zuordnungen der Form:

$$\text{Klartextbegriff} \Longleftrightarrow \text{Codebegriff}$$

aufgelistet sind. Bei der Verschlüsselung mit einem Codebuch wird einfach der Klartextbegriff durch den Chiffrebegriff ersetzt. Um zu Dechiffrieren, muß der Empfänger der Nachricht ebenfalls das Codebook besitzen. Der Empfänger schlägt dann einfach den Chiffrebgriff im Codebuch nach und ersetzt ihn durch den zugeordneten Klartextbegriff.

Wird der mit einem Codebook chiffrierte Text dann anschließend mit Hilfe der Shift-Chiffre ein weiteres Mal verschlüsselt, erzielt man eine höhere Sicherheit.

Claude E. Shannon hat diese Idee in seiner Arbeit [199] formalisiert und das Konzept der **Produkt Chiffren** eingeführt. Dieses Konzept spielt in modernen symmetrischen Kryptosystemen wie der Data Encryption Standard oder der Advanced Encryption Standard eine fundamentale Rolle.

Der Einfachheit halber sehen wir uns Kryptosysteme an, bei denen der Klartextraum gleich dem Chiffretextraum ist. Solche Systeme heißen *endomorphische Kryptosysteme*. Solch eine System ist also ein Fünf-Tupel

$$\mathcal{K} = (P, P, K, E, D)$$

mit $P = C$, i.e. der Klartextraum ist identisch mit dem Chiffretextraum, K ist der Schlüsselraum, E ist die Menge der Verschlüsselungsalgorithmen und D die Menge der Dechiffrieralgorithmen. Seien

$$\mathcal{K}_1 = (P, P, K_1, E_1, D_1)$$
$$\mathcal{K}_2 = (P, P, K_2, E_2, D_2)$$

zwei endomorpische Kryptosysteme, die den gleichen Klar- bzw. Chiffretextraum haben. Dann ist das **Produkt Kryptosystem** definiert als

$$\mathcal{K}_1 \times \mathcal{K}_2 = (P, P, K_1 \times K_2, E, D)$$

Ein Schlüssel des Produkt Kryptosystems hat die Form:

$$\overline{k} = (k_1, k_2) \ \in \ K_1 \times K_2.$$

Die Verschlüsselung im Produktsystem ist folgendermaßen definiert:

Für jeden Schlüssel $\overline{k} = (k_1, k_2)$ ist die Chiffrierung definiert durch

$$e_{\overline{k}}(x) = e_{(k_1,k_2)}(x) = e_{k_2}\left(e_{k_1}(x)\right)$$

Die Dechiffrierung geschieht durch

$$d_{\overline{k}}(x) = d_{(k_1,k_2)}(x) = d_{k_1}\big(d_{k_2}(x)\big)$$

Mit anderen Worten, der Klartext x wird zunächst mit e_{k_1} verschlüsselt, das entstehende Chiffrat wird anschließend mit e_{k_2} nochmals chiffriert. Die Dechiffrierung ist ähnlich, man beachte jedoch, dass die Reihenfolge umgedreht ist:

$$\begin{aligned} d_{\overline{k}}\big(e_{\overline{k}}(x)\big) &= d_{(k_1,k_2)}\big(e_{(k_1,k_2)}(x)\big) \\ &= d_{(k_1,k_2)}\big(e_{k_2}(e_{k_1}(x)\big) \\ &= d_{k_1}\Big(g_{k_2}\big(e_{k_2}(e_{k_1}(x))\big)\Big) \\ &= d_{k_1}(e_{k_1}(x)) \\ &= x \end{aligned}$$

Beispiel [2.8]

Wir definieren zunächst die **Multiplikationschiffre** wie folgt:

Sei $P = C = \mathbb{Z}_{26}$ und sei der Schlüsselraum

$$K = \mathbb{Z}_{26}^* = \{a \in \mathbb{Z}_{26} \mid \mathrm{ggT}(a, 26) = 1\}$$

Dann definieren wir die Verschlüsselung e_a für einen festen Wert $a \in K$ durch:

$$e_a(x) = a \cdot x \bmod 26$$

Die Dechiffrierung ist definiert über

$$d_a(y) = a^{-1} \cdot y \bmod 26$$

Hier ist $x, y \in P = \mathbb{Z}_{26}$.

Wir betrachten nun das Produkt Kryptosystem, das aus der oben definierten Multiplikationschiffre und der Shift Chiffre aus Abschnitt [2.2] besteht. Wir betrachten daher die Multiplikationschiffre

$$\mathcal{M} = (P, P, K_1, E, D)$$

mit den oben definierten Parametern und die Shift Chiffre

$$\mathcal{S} = (P, P, K_2, E, D)$$

wobei ein Schlüssel aus $\mathcal{S}$ eine Zahl $k \in \mathbb{Z}_{26}$ ist. Die Verschlüsselung ist durch die Vorschrift

$$e_k(x) = (x - k) \bmod 26$$

gegeben.

Ein Schlüssel in $\mathcal{M}$ ist ein Element $a \in \mathbb{Z}_{26}$ mit $\mathtt{ggT}(a, 26) = 1$. Die Verschlüsselungsvorschrift ist:

$$e_a(x) = a \cdot x \bmod 26$$

Daher ist ein Schlüssel des Produktsystems $\mathcal{M} \times \mathcal{S}$ ein Paar (a, k) und es gilt die Verschlüsselungsvorschrift

$$e_{(a,k)}(x) = (a \cdot x + k) \bmod 26$$

i.e. dies ist die Verschlüsselungsvorschrift der Affinen Chiffre.

Es kann leicht gezeigt werden, dass gilt

$$\mathcal{M} \times \mathcal{S} = \mathcal{S} \times \mathcal{M}$$

i.e. das Produktsystem ist kommutativ.

Kryptosysteme sind weiterhin dadurch charakterisiert, dass der Schlüsselraum einer bestimmten Wahrscheinlichkeitsverteilung unterliegt. So ist die Wahrscheinlichkeit eines Keys in der Shift Chiffre $P(k_1) = 1/26$, *i.e.* die 26 mögliche Schlüssel sind gleichverteilt. Gleiches gilt natürlich auch für die Verteilung der Schlüssel der Multiplikationschiffre. Da es 12 coprime Zahlen zu 26 gibt, ist die Wahrscheinlichkeit für das Auftreten eines Schlüssels $P(k_2) = 1/12$. Bei der Produktchiffre definiert man in natürlicher Weise die Wahrscheinlichkeit des Auftretens eines Schlüsselwertes als Produkt

$$P\big[(k_1, k_2)\big] = P(k_1) \times P(k_2)$$

Daher ist die Wahrscheinlichkeit der Wahl eines Schlüssels der Produktchiffre $\mathcal{M} \times \mathcal{S}$

$$P\big[(k_1, k_2)\big] = P(k_1) \times P(k_2) = \frac{1}{26} \times \frac{1}{12} = \frac{1}{312}$$

was der Schlüsselverteilung der affinen Chiffre entspricht.

2.10 One–Time Pad

Die Untersuchung des Grades der Sicherheit, die ein Kryptosystem bietet, kann — da man unendliche Rechen– und Speicherkapazitäten zulässt — nicht mit Methoden der Komplexitätstheorie erfasst werden. Der geeignete Rahmen, in dem diese Fragen diskutiert werden können, ist die Wahrscheinlichkeits– und Informationstheorie. Dies geht auf Arbeiten von Claude Shannon zurück [198–200], der in [199] den informationstheoretischen Hintergrund von Kryptosystemen untersuchte.

In diesem Abschnitt betrachten wir ein Kryptosystem $\mathcal{K} = (P, C, K, E, D)$ mit folgenden Eigenschaften

- P Klartextraum
- C Chiffretextraum
- K Schlüsselraum
- E Verschlüsselungsalgorithmus
- D Dechiffrieralgorithmus.

Ein Schlüssel $k \in K$ wird nur für eine einzige Verschlüsselung verwendet. Wir nehmen nun an, es gibt eine Wahrscheinlichkeitsverteilung auf dem Klartextraum P. Daher definiert das Klartextraumelement — sprich die Zeichen aus denen der Klartext aufgebaut wird — eine Zufallsvariable, die wir mit $\mathbf{x}$ bezeichnen. Wir bezeichnen die *a priori* Wahrscheinlichkeit für das Ereignis, dass das Klartextzeichen x auftritt mit

$$P[\mathbf{x} = x].$$

Wir nehmen ebenfalls an, dass der Schlüssel k von Alice und Bob zufällig gewählt wird, wobei eine festgelegte Wahrscheinlichkeitsverteilung genutzt wird. Üblicherweise sind alle Schlüsselwerte gleich wahrscheinlich, daher unterliegt der Verteilung der Keys eine Gleichverteilung. Dies muss aber nicht immer der Fall sein. Der Schlüssel definiert daher eine weitere Zufallsvariable, die wir mit $\mathbf{K}$ bezeichnen. Die Wahrscheinlichkeit für das Eintreten des Ereignisses, dass der konkrete Schlüssel k gewählt wird, bezeichnen wir mit

$$P[\mathbf{K} = k].$$

Üblicherweise wird der Schlüssel gewählt, bevor bekannt ist, welcher Klartext verschlüsselt werden soll. Daher kann man die vernünftige Annahme machen, dass der Schlüssel und der Klartext unabhängige Zufallsvariablen sind.

Die beiden Wahrscheinlichkeitsverteilungen auf dem Klartextraum P und dem Schlüsselraum K induzieren eine Wahrscheinlichkeitsverteilung auf dem Chiffreraum C. Wir können daher auch die Chiffretextelemente als Zufallsvariable betrachten, diese bezeichnen wir mit $\mathbf{y}$. Es ist nicht schwierig die Wahrscheinlichkeit $P[\mathbf{y} = y]$ zu berechnen, dass y der übertragene Chiffretext ist. $P[\mathbf{y} = y]$ ist also die Wahrscheinlichkeit für das Ereignis, dass das Chiffretextzeichen y

auftritt, wenn die Ereigniss $\mathbf{K} = k$ und $\mathbf{x} = x$ eintreten. Für einen Schlüssel $k \in K$ definieren wir:

$$C(k) = \{e_k(x) \mid x \in P\} = \{y \in C \mid y = e_k(x), x \in P\}$$

i.e. $C(k)$ ist die Menge der möglichen Chiffretexte falls k der Schlüssel ist. Dann gilt für jedes $y \in C$:

$$P[\mathbf{y} = y] = \sum_{\{k:y \in C(k))\}} P[\mathbf{K} = k] \cdot P[\mathbf{x} = d_k(y)]$$

Weiterhin können wir für jedes $y \in C$ und $x \in P$ die **bedingte Wahrscheinlichkeit** $P[\mathbf{y} = y \mid \mathbf{x} = x]$ berechnen. $P[\mathbf{y} = y \mid \mathbf{x} = x]$ beschreibt die Wahrscheinlichkeit, dass y das Chiffretextzeichen ist, wenn das Klartextzeichen x aufgetreten ist. Es folgt:

$$P[\mathbf{y} = y \mid \mathbf{x} = x] = \sum_{\{k:x = d_k(y)\}} P[\mathbf{K} = k].$$

Damit kann nun die *bedingte Wahrscheinlichkeit* $P[\mathbf{x} = x \mid \mathbf{y} = y]$ berechnet werden. Diese beschreibt die Wahrscheinlichkeit für x, wenn der Chiffretext y gegeben ist. Mit anderen Worten, die bedingte Wahrscheinlichkeit $P[\mathbf{x} = x \mid \mathbf{y} = y]$ ist ein Maß dafür, welche Informationen ein Angreifer – dieser kann ja den Chiffretext y lesen – über den Klartext erhält. Diese Wahrscheinlichkeit nennt man auch *a posteriori* Wahrscheinlichkeit. Mit Hilfe des Satzes von BAYES (siehe [79, 178, 189]) erhalten wir:

$$\begin{aligned} P[\mathbf{x} = x \mid \mathbf{y} = y] &= \frac{P[\mathbf{x} = x] \times \sum_{\{K:x = d_k(y)\}} P[\mathbf{K} = k]}{\sum_{\{k:y \in C(k)\}} P[\mathbf{K} = k] P[\mathbf{x} = d_k(y)]} \\ &= \frac{P[\mathbf{x} = x] \times \sum_{\{K:x = d_k(y)\}} P[\mathbf{K} = k]}{P[\mathbf{y} = y]} \end{aligned}$$

Beispiel [2.9]

Wir demonstrieren die Begriffe dieses Abschnitts anhand eines einfachen Beispiels. Sei

$$P = \{0, 1\}$$

der Klartextraum mit

$$P[\mathbf{x} = 0] = \frac{1}{4} \quad \text{und} \quad P[\mathbf{x} = 1] = \frac{3}{4}$$

Der Schlüsselraum besteht aus den drei Schlüsselwerten

$$K = \{k_1, k_2, k_3\}$$

mit der Verteilung

$$\begin{aligned} P[\mathbf{K} = k_1] &= \frac{1}{2} \\ P[\mathbf{K} = k_2] &= \frac{1}{4} \\ P[\mathbf{K} = k_3] &= \frac{1}{4} \end{aligned}$$

Der Chiffretextraum besteht aus vier Elementen

$$C = \{a, b, c, d\}$$

Die Verschlüsselungsfunktionen sind definiert über:

$$\begin{aligned} e_{k_1}(0) &= a \\ e_{k_1}(1) &= b \\ e_{k_2}(0) &= b \\ e_{k_2}(1) &= c \\ e_{k_3}(0) &= c \\ e_{k_3}(1) &= d \end{aligned}$$

Damit erhalten wir auch die Dechiffrierfunktionen zu:

$$\begin{aligned} d_{k_1}(a) &= 0 \\ d_{k_1}(b) &= 1 \\ d_{k_2}(b) &= 0 \\ d_{k_2}(c) &= 1 \\ d_{k_3}(c) &= 0 \\ d_{k_3}(d) &= 1 \end{aligned}$$

Mit diesen Werten können wir die induzierte Wahrscheinlichkeitsverteilung auf dem Chiffreraum C berechnen. Zunächst haben wir:

$$\begin{aligned} C(k_1) &= \{e_{k_1}(x) \mid x \in P\} = \{a, b\} \\ C(k_2) &= \{e_{k_2}(x) \mid x \in P\} = \{b, c\} \\ C(k_3) &= \{e_{k_3}(x) \mid x \in P\} = \{c, d\} \end{aligned}$$

Also, $C(k_1)$ ist die Menge der möglichen Chiffretextzeichen, falls der Schlüssel k_1 benutzt wird, usw. Dann ist:

$$\begin{aligned} P[\mathbf{y} = a] &= \sum_{\{k: a \in C(k))\}} P[\mathbf{K} = k] P[\mathbf{x} = d_k(y)] \\ &= P[\mathbf{K} = k_1] P[\mathbf{x} = d_{k_1}(a)] + P[\mathbf{K} = k_2] P[\mathbf{x} = d_{k_2}(a)] \\ &\quad + P[\mathbf{K} = k_3] P[\mathbf{x} = d_{k_3}(a)] \\ &= P[\mathbf{K} = k_1] P[\mathbf{x} = d_{k_1}(a)] \\ &= \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8} \end{aligned}$$

denn:

$$\begin{aligned} P[\mathbf{x} = d_{k_2}(a)] &= 0 \\ P[\mathbf{x} = d_{k_3}(a)] &= 0 \end{aligned}$$

i.e. die Wahrscheinlichkeit, dass irgendein Klartextzeichen aufgetreten ist, wenn das Chiffretextzeichen a vorliegt und die Schlüssel k_2 oder k_3 zum Chiffrieren verwendet wird, ist 0.

Weiterhin folgt:

$$\begin{aligned} P[\mathbf{y}=b] &= \sum_{\{k:b\in C(k))\}} P[\mathbf{K}=k]P[\mathbf{x}=d_k(b)] \\ &= P[\mathbf{K}=k_1]P[\mathbf{x}=d_k(b)] + P[\mathbf{K}=k_2]P[\mathbf{x}=d_k(b)] \\ &= \frac{1}{2}\cdot\frac{3}{4}+\frac{1}{4}\cdot\frac{1}{4} \\ &= \frac{7}{16} \end{aligned}$$

$$\begin{aligned} P[\mathbf{y}=c] &= \sum_{\{k:c\in C(k))\}} P[\mathbf{K}=k]P[\mathbf{x}=d_k(c)] \\ &= P[\mathbf{K}=k_2]P[\mathbf{x}=d_k(c)] + P[\mathbf{K}=k_3]P[\mathbf{x}=d_k(c)] \\ &= \frac{1}{4}\cdot\frac{3}{4}+\frac{1}{4}\cdot\frac{1}{4} \\ &= \frac{1}{4} \end{aligned}$$

$$\begin{aligned} P[\mathbf{y}=d] &= \sum_{\{k:d\in C(k))\}} P[\mathbf{K}=k]P[\mathbf{x}=d_k(d)] \\ &= P[\mathbf{K}=k_3]P[\mathbf{x}=d_k(d)] \\ &= \frac{1}{4}\cdot\frac{3}{4} = \frac{3}{16} \end{aligned}$$

Damit können nun die bedingten Wahrscheinlichkeiten berechnet werden:

$$\begin{aligned} P[\mathbf{x}=0 \mid \mathbf{y}=a] &= \frac{P[\mathbf{x}=0]\times\sum_{\{K:0=d_k(a)\}}P[\mathbf{K}=k]}{P[\mathbf{y}=a]} \\ &= \frac{\frac{1}{4}\cdot\frac{1}{2}}{\frac{1}{8}} = 1 \end{aligned}$$

Dieses Ergebnis ist nichts anderes als folgende Aussage: Wenn das Chiffretextzeichen a gelesen wird, dann kann – unter der gegebenen Chiffre – das Klartextzeichen nur eine 0 sein. Da die bedingte Wahrscheinlichkeit 1 ist, ist dies mit Gewissheit der Fall.

$$\begin{aligned} P[\mathbf{x}=0 \mid \mathbf{y}=b] &= \frac{P[\mathbf{x}=0]\times\sum_{\{K:0=d_k(b)\}}P[\mathbf{K}=k]}{P[\mathbf{y}=b]} \\ &= \frac{P[\mathbf{x}=0]\times P[\mathbf{K}=k_2]}{P[\mathbf{y}=b]} \\ &= \frac{\frac{1}{4}\cdot\frac{1}{4}}{\frac{7}{16}} = \frac{1}{7} \end{aligned}$$

$$P[\mathbf{x}=0 \mid \mathbf{y}=c] = \frac{P[\mathbf{x}=0] \times \sum_{\{K:0=d_k(c)\}} P[\mathbf{K}=k]}{P[\mathbf{y}=c]}$$
$$= \frac{P[\mathbf{x}=0] \times P[\mathbf{K}=k_3]}{P[\mathbf{y}=c]}$$
$$= \frac{\frac{1}{4} \cdot \frac{1}{4}}{\frac{1}{4}} = \frac{1}{4}$$

$$P[\mathbf{x}=0 \mid \mathbf{y}=d] = \frac{P[\mathbf{x}=0] \times \sum_{\{K:0=d_k(d)\}} P[\mathbf{K}=k]}{P[\mathbf{y}=d]}$$
$$= 0$$

$$P[\mathbf{x}=1 \mid \mathbf{y}=a] = \frac{P[\mathbf{x}=1] \times \sum_{\{K:1=d_k(a)\}} P[\mathbf{K}=k]}{P[\mathbf{y}=a]}$$
$$= 0$$

$$P[\mathbf{x}=1 \mid \mathbf{y}=b] = \frac{P[\mathbf{x}=1] \times \sum_{\{K:1=d_k(b)\}} P[\mathbf{K}=k]}{P[\mathbf{y}=b]}$$
$$= \frac{P[\mathbf{x}=1] \times P[\mathbf{K}=k_1]}{P[\mathbf{y}=b]}$$
$$= \frac{\frac{3}{4} \cdot \frac{1}{2}}{\frac{7}{16}} = \frac{6}{7}$$

$$P[\mathbf{x}=1 \mid \mathbf{y}=c] = \frac{P[\mathbf{x}=1] \times \sum_{\{K:1=d_k(c)\}} P[\mathbf{K}=k]}{P[\mathbf{y}=c]}$$
$$= \frac{P[\mathbf{x}=1] \times P[\mathbf{K}=k_2]}{P[\mathbf{y}=c]}$$
$$= \frac{\frac{3}{4} \cdot \frac{1}{4}}{\frac{1}{4}} = \frac{3}{4}$$

Schließlich:

$$P[\mathbf{x}=1 \mid \mathbf{y}=d] = \frac{P[\mathbf{x}=1] \times \sum_{\{K:1=d_k(d)\}} P[\mathbf{K}=k]}{P[\mathbf{y}=d]}$$
$$= \frac{P[\mathbf{x}=1] \times P[\mathbf{K}=k_3]}{P[\mathbf{y}=d]}$$
$$= \frac{\frac{3}{4} \cdot \frac{1}{4}}{\frac{3}{16}} = 1.$$

Wir können nun das Konzept der perfekten Sicherheit definieren. Qualitativ bedeutet die perfekte Sicherheit, dass die Angreiferin Eve keinerlei Informationen über den Klartext erhält, wenn sie den Chiffretext untersucht.

Definition [2.9]:

Ein Kryptosystem hat die Eigenschaft der **perfekten Sicherheit**, wenn gilt:

$$P[\mathbf{x} = x \mid \mathbf{y} = y] = P[\mathbf{x} = x] \quad \forall\, x \in P, y \in C.$$

Das bedeutet, die *a posteriori* Wahrscheinlichkeit, dass der Klartext x auftritt, wenn der Chiffretext y gelesen wird ist gleich der *a priori* Wahrscheinlichkeit von x.

Wir zeigen nun, dass die Shift Chiffre perfekte Sicherheit bietet. Intutitiv kann das folgendermaßen eingesehen werden. Wenn irgendein Chiffretextzeichen $y \in \mathbb{Z}_{26}$ gegeben ist, dann kann jedes Klartextzeichen $x \in \mathbb{Z}_{26}$ eine mögliche Entschlüsselung von y sein. Dies hängt natürlich von dem gewählten Schlüssel ab.

Theorem [1]:

Angenommen die 26 Schlüssel der Shift Chiffre werden gleichverteilt eingesetzt, *i.e.* mit Wahrscheinlichkeit $\frac{1}{26}$. Für eine beliebige Wahrscheinlichkeitsverteilung des Klartextes liefert die Shift Chiffre perfekte Sicherheit.

Beweis:

Die Shift Chiffre ist durch folgenden Satz von Parametern definiert:

$$P = C = K = \mathbb{Z}_{26}$$

Für einen Schlüssel $k \in \mathbb{Z}_{26}$ lautet die Verschlüsselung e_k:

$$e_k(x) = (x + k) \bmod 26$$

und die Dechiffrierfunktion ist:

$$d_k(y) = (y - k) \bmod 26$$

Damit können wir die induzierte Wahrscheinlichkeitsverteilung auf dem Chif-

fretextraum C wie folgt berechnen:

$$\begin{aligned} P[\mathbf{y} = y] &= \sum_{k \in \mathbb{Z}_{26}} P[\mathbf{K} = k] \cdot P[\mathbf{x} = d_k(y)] \\ &= \sum_{k \in \mathbb{Z}_{26}} \frac{1}{26} \cdot P[\mathbf{x} = y - k] \\ &= \frac{1}{26} \cdot \sum_{k \in \mathbb{Z}_{26}} P[\mathbf{x} = y - k]. \end{aligned}$$

Für einen festen Wert y bilden die 26 Werte $(y - k) \bmod 26$ eine Permutation von $\mathbb{Z}_{26}$. Daher ist:

$$\begin{aligned} \sum_{k \in \mathbb{Z}_{26}} P[\mathbf{x} = y - k] &= \sum_{x \in \mathbb{Z}_{26}} P[\mathbf{x} = x] \\ &= 1. \end{aligned}$$

Daher ist die auf C induzierte Wahrscheinlichkeitsverteilung:

$$P[\mathbf{y} = y] = \frac{1}{26}.$$

Da für jedes Klartext-Chiffretextpaar x, y der eindeutige Schlüssel k wegen $e_k(x) = y$ durch $k = (y - x) \bmod 26$ gegeben ist, folgt weiterhin für die bedingte Wahrscheinlichkeit, dass y auftritt, wenn der Klartextwert x angenommen wird:

$$\begin{aligned} P[\mathbf{y} = y \mid \mathbf{x} = x] &= P[\mathbf{K} = (y - x) \bmod 26] \\ &= \frac{1}{26} \end{aligned}$$

Wendet man nun den Satz von BAYES an, folgt:

$$\begin{aligned} P[\mathbf{x} = x \mid \mathbf{y} = y] &= \frac{P[\mathbf{x} = x] \cdot P[\mathbf{y} = y \mid \mathbf{x} = x]}{P[\mathbf{y} = y} \\ &= \frac{P[\mathbf{x} = x] \cdot \frac{1}{26}}{\frac{1}{26}} \\ &= P[\mathbf{x} = x] \end{aligned}$$

Daher liegt perfekte Sicherheit vor.

Damit haben wir gezeigt, dass die Shift Chiffre nicht brechbar ist — *vorausgesetzt, bei der Verschlüsselung jedes Klartextzeichens wird ein neuer zufälliger Schlüssel genutzt.*

Betrachten wir die perfekte Sicherheit noch etwas genauer. Mit Hilfe des Satzes von BAYES ist ableitbar, dass die Bedingung

$$P[\mathbf{x} = x \mid \mathbf{y} = y] = P[\mathbf{x} = x] \quad \forall\, x \in P, y \in C$$

gleichbedeutend ist zu

$$P[\mathbf{y} = y \mid \mathbf{x} = x] = P[\mathbf{y} = y] \quad \forall\, x \in P, y \in C.$$

Wir machen nun die Annahme, dass $P[\mathbf{y} = y] > 0$ für alle $y \in C$. Dies ist durchaus vernünftig, da C auf die Chiffretexte eingeschränkt werden kann, für die $P > 0$ ist. $P[\mathbf{y} = y] = 0$ heißt ja gerade, dass der Chiffretext y nicht verwendet wird. Dann betrachten wir ein $x \in P$. Für jedes $y \in C$ haben wir

$$P[\mathbf{y} = y \mid \mathbf{x} = x] = P[\mathbf{y} = y] > 0.$$

Für jedes y muss es mindestens einen Schlüssel k geben mit $e_k(x) = y$. Damit muss aber $\mid K \mid \geq \mid C \mid$ gelten. In jedem Kryptosystem muss weiterhin immer gelten $\mid C \mid \geq \mid P \mid$, da jede Verschlüsselungsfunktion aufgrund der Eindeutigkeit und Umkehrbarkeit injektiv sein muß. Im Spezialfall, wenn alle Mengen die gleiche Mächtigkeit haben, *i.e.* $\mid P \mid = \mid C \mid = \mid K \mid$ kann eine Charakterisierung angegeben werden, wann das Kryptosystem perfekte Sicherheit liefert. Der folgende Satz geht auf Claude Shannon zurück [199].

Theorem [2]:

Angenommen

$$\mathcal{K} = (P, C, K, E, D)$$

ist ein Kryptosystem mit der Eigenschaft

$$\mid P \mid = \mid C \mid = \mid K \mid .$$

Das Kryptosystem $\mathcal{K}$ liefert perfekte Sicherheit dann und nur dann, wenn jeder Schlüssel mit gleicher Wahrscheinlichkeit $\frac{1}{|K|}$ genutzt wird und für jedes $x \in P$ und jedes $y \in C$ gibt es einen eindeutigen Schlüssel $k \in K$ mit $e_k(x) = y$.

Beweis:

Wir haben zwei Richtungen zu zeigen:

① Ein Kryptosystem mit $\mid P \mid = \mid C \mid = \mid K \mid$ liefert perfekte Sicherheit $\Longrightarrow$

(a) jeder Schlüssel wird mit gleicher Wahrscheinlichkeit $1/\mid K \mid$ benutzt

(b) für jedes $x \in P$ und jedes $y \in C$ existiert genau ein Schlüssel $k \in K$ mit $e_k(x) = y$

Ad a Setze $n = \mid K \mid$. Sei $P = \{x_i \mid 1 \leq i \leq n\}$ und $y \in C$ fest. Wir können die n Schlüssel mit $k_1, k_2, k_3, \ldots, k_n$ derart bezeichnen, daß gilt:

$$e_{k_i}(x_i) = y. \qquad 1 \leq i \leq n$$

Mit Hilfe des Satzes von BAYES folgt:

$$\begin{aligned} P[\mathbf{x} = x_i \mid \mathbf{y} = y] &= \frac{P[\mathbf{y} = y \mid \mathbf{x_i} = x] \cdot P[\mathbf{x} = x_i]}{P[\mathbf{y} = y]} \\ &= \frac{P[\mathbf{K} = k_i] \cdot P[\mathbf{x} = x_i]}{P[\mathbf{y} = y]} \\ &\stackrel{!}{=} P[\mathbf{x} = x_i] \end{aligned}$$

denn es gilt nach Voraussetzung die Bedingung für die perfekte Sicherheit

$$P[\mathbf{x} = x_i \mid \mathbf{y} = y] = P[\mathbf{x} = x_i]$$

Daher folgt:

$$P[\mathbf{K} = k_i] = P[\mathbf{y} = y] \quad i = 1, 2, 3, \ldots, n$$

Dies besagt, dass die Schlüssel K mit gleicher Wahrscheinlichkeit $P[\mathbf{y} = y]$ verwendet werden. Da die Anzahl der Schlüssel aber $\mid K \mid$ ist, muss gelten:

$$P[\mathbf{K} = k_i] = \frac{1}{\mid K \mid}.$$

Ad b Für jedes $x \in P$ und $y \in C$ muß es mindestens einen Schlüssel $k \in K$ geben mit $e_k(x) = y$. Daher gilt die Ungleichung:

$$\begin{aligned} \mid C \mid &= \mid \{e_k(x) \mid k \in K\} \mid \\ &\leq \mid K \mid \end{aligned}$$

Da wir jedoch annehmen, daß $\mid C \mid = \mid K \mid$, gilt:

$$\mid \{e_k(x) \mid k \in K\} \mid = \mid K \mid$$

Das heißt, es gibt keine zwei verschiedene Schlüssel $k_1, k_2 \in K$ mit

$$e_{k_1}(x) = e_{k_2}(x) = y.$$

Damit ist gezeigt: Für jedes $x \in P$ und $y \in C$ existiert genau ein $k \in K$ mit $e_k(x) = y$.

② Umkehrung: Angenommen, die beiden Bedingungen

(a) jeder Schlüssel wird mit gleicher Wahrscheinlichkeit $1/\mid K \mid$ benutzt

(b) für jedes $x \in P$ und jedes $y \in C$ existiert genau ein Schlüssel $k \in K$ mit $e_k(x) = y$

sind erfüllt. $\Longrightarrow$ Das Kryptosystem liefert perfekte Sicherheit.

Eine bekannte Realisierung der perfekten Sicherheit ist die **One-Time Pad** Verschlüsselung, die von GILBERT VERNAM bei AT & T im Jahre 1917 beschrieben (siehe [118, pp. 395-403]) und von Major JOSEPH O. MAUBORGNE

eingesetzt wurde. Interessant dabei ist, dass die One-Time Pad Verschlüsselung lange als nicht brechbares Kryptosystem angesehen wurde. Erst etwa 30 Jahre später konnte CLAUDE SHANNON in der Arbeit [199] zeigen, dass der One-Time Pad in der Tat absolute Sicherheit bietet.

Definition [2.10]:

Sei $n \geq 1$ eine ganze Zahl und sei

$$P = C = K = (\mathbb{Z}_2)^n.$$

Für $k \in \mathbb{Z}_2^n$ definieren wir die Verschlüsselungsfunktion $e_k(x)$ als die Vektorsumme modulo 2 von x und k. Äquivalent ist, man bildet die `XOR` Verknüpfung der beiden Bitstrings. Sei also der Klartext

$$x = (x_1, x_2, \ldots, x_n) \in (\mathbb{Z}_2)^n$$

und der Schlüssel

$$k = (k_1, k_2, \ldots, k_n) \in (\mathbb{Z}_2)^n$$

gegeben. Dann ist

$$e_k(x) = (x_1 + k_1, x_2 + k_2, \ldots, x_n + k_n) \bmod 2$$

die Verschlüsselung. Die Entschlüsselung ist identisch zu der Verschlüsselung. Falls $y = (y_1, y_2, \ldots, y_n) \in (\mathbb{Z}_2)^n$, dann ist

$$d_k(y) = (y_1 + k_1, y_2 + k_2, \ldots, y_n + k_n) \bmod 2.$$

Mit Hilfe des Satzes von SHANNON ist leicht einzusehen, dass die One–Time Pad Verschlüsselung eine perfekte Sicherheit bietet. Das System ist weiterhin dadurch attraktiv, dass die Ver– und Entschlüsselung sehr einfach ist. VERNAM patentierte seine Idee mit der Hoffung, dass sich das One–Time Pad für kommerzielle Zwecke verbreiten würde — damals zur Verschlüsselung von telegraphischen Nachrichten. Unglücklicherweise haben Kryptosysteme mit perfekter Sicherheit einige gravierende Nachteile, die ein Einsatz in der Praxis sehr erschwert. Die oben (implizit) formulierte Bedingung $| K | \geq | P |$ bedeutet im Klartext, dass die Menge der Schlüssel, die zwischen zwei Parteien auf sicherem Weg ausgetauscht werden muss, mindestens so groß ist wie die Klartextmenge. Beispielsweise benötigt man zur Anwendung der One–Time Pad Verschlüsselung n Schlüsselbits um n Klartextbits zu chiffrieren. Dies wäre kein großes Problem, wenn der gleiche Schlüssel benutzt werden könnte, um mehrere Klartextpassagen zu verschlüsseln. *Die Sicherheit eines uneingeschränkt sicheren Kryptosystems hängt jedoch entscheidend davon ab, dass jeder Schlüssel genau für eine einzige Verschlüsselung genutzt wird.*[9]

[9] Daher stammt die Bezeichnung One-Time Pad.

Der One–Time Pad ist anfällig gegenüber einer Known Plaintext Attacke, da der Schlüssel k durch eine `XOR` Verknüpfung der Bitstrings x und $e_k(x)$ berechnet werden kann. Daher muss für jede Nachricht, die mit dem One–Time Pad verschlüsselt wird, ein neuer Schlüssel generiert werden und auf sichere Weise zum Empfänger gebracht werden. Dies erzeugt natürlich ein ernstzunehemdenes Schlüsselmanagementproblem, das den Einsatz dieses Verfahrens in der Praxis sehr stark einschränkt.

Klassisch ist ein One–Time Pad also nichts anderes als eine große, sich niemals wiederholende Menge von echt zufälligen Schlüsselzeichen, die auf Stapeln von Papierblättern niedergeschrieben sind. Dieser Stapel von zufälligen Schlüsselzeichen nennt man **Pad** (Block). Der Sender benutzt ein Schlüsselzeichen nach dem anderen aus dem Pad um jeweils genau ein Klartextzeichen zu verschlüsseln. Die Verschlüsselung besteht aus der Addition modulo 26 des Klartextzeichens mit dem Schlüsselzeichen.

Jedes Schlüsselzeichen wird genau einmal für genau eine Nachricht benutzt. Der Sender verschlüsselt auf diese Weise die komplette Nachricht und vernichtet die verwendeten Seiten des Pads. Der Empfänger hat ein identisches Pad, benutzt jedes Schlüsselzeichen des Pads für die Dechiffrierung des Chiffretextes (Zeichen für Zeichen) und vernichtet anschließend die gleichen Seiten des Blocks wie der Sender.

Beispiel [2.10]

Der Klartext, der mit dem One–Time Pad verschlüsselt werden soll ist

`onetimepad`

Die Sequenz der Schlüsselzeichen aus dem Pad sei (dies müssen gleich viele — also 10 — Zeichen sein):

`tbfrgfarfm`

Damit berechnet der Sender den Chiffretext durch Addition modulo 26 zu:

`onetimepad`	$\Longrightarrow$	14 13 04 19 08 12 04 15 00 03
`tbfrgfarfm`	$\Longrightarrow$	19 01 05 17 06 05 00 17 05 12
Addition mod 26		
`IOJKOREGFP`	$\Longleftarrow$	07 14 09 10 14 17 04 06 05 15

Angenommen, ein Angreifer hat keinen Zugang zu dem One–Time Pad, dann ist dieses Verfahren absolut sicher. Ein vorliegender Chiffretext kann mit gleicher Wahrscheinlichkeit aus irgendeinem Klartext mit gleicher Länge erzeugt worden sein. Mit gleicher Wahrscheinlichkeit könnte die Zeichenfolge für den Schlüssel die Sequenz

`goxkbexcxd`

sein. Dies führt bei gleichem Chiffretext auf den Klartext

`bamannheim`

Nochmals zur Klarstellung:

> Da jede Klartextnachricht gleich wahrscheinlich ist, besteht keine Möglichkeit für den Kryptoanalyst herauszufinden, welche Klartextnachricht die korrekte ist. Eine zufällige Sequenz von Schlüsselzeichen, die zu einer nicht-zufälligen Klartextnachricht addiert wird, erzeugt eine völlig zufällige Chiffretextnachricht. Daran ändert auch auch eine beliebig zur Verfügung stehende Rechenleistung nichts!

Die Sicherheit der One–Time Pad Verschlüsselung hängt entscheidend von zwei Faktoren ab:

① Die Schlüsselzeichen müssen echt zufällig sein. Jede Attacke gegen dieses Kryptosystem richtet sich gegen das Verfahren, wie die Schlüsselzeichenfolge generiert wird. Ein Pseudozufallszahlengenerator ist hierfür nicht ausreichend. Um absolute Sicherheit zu gewährleisten, müssen echte Zufallszahlen verwendet werden.

② Der zweite Punkt ist, dass jede Schlüsselsequenz genau einmal verwendet werden darf. Selbst wenn ein Pad der Größe von mehreren Gigabytes verwendet wird, darf eine Schlüsselsequenz niemals zwei Mal verwendet werden. Falls ein Kryptoanalyst mehrere Chiffretexte zur Verfügung hat, deren Schlüssel überlappen, kann der Klartext rekonstruiert werden. Der Kryptanalyst verschiebt Paare von vorliegenden Chiffretexten gegeneinander und zählt die Übereinstimmungen an jeder Position. Falls die Verschiebung die richtige Position hat, geht die Anzahl der Übereinstimmung sprunghaft in die Höhe. Dies hängt von der Statistik der Sprache des Klartextes ab. Von diesem Punkt an ist die Kryptoanalyse einfach.

One–Time Pads werden in der Praxis nur für ultra–sichere Übertragungskanäle genutzt. Man geht davon aus, dass der heiße Draht zwischen Washington und Moskau während des Kalten Krieges mit einem One–Time Pad verschlüsselt war [193, pp. 15-17].

2.11 Rotor Maschinen

In den 20er Jahren des letzten Jahrhunderts wurden eine Reihe mechanischer

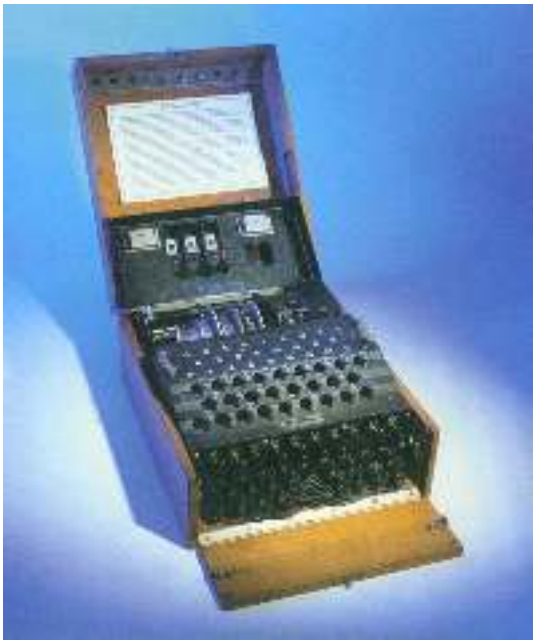


Abbildung 2.4: Die Enigma (aus [191]).

— und elektromechanischer — Verschlüsselungsgeräte entwickelt, um den Verschlüsselungsprozess insbesondere in der aufkommenden Telegraphietechnik zu automatisieren. Die meisten dieser Geräte beruhten auf dem Rotor–Prinzip. Ein Rotor ist ein Rad, das so verdrahtet ist, dass eine Substitution eines Zeichens durchgeführt wird.

Die sicherlich berühmteste Rotormaschine ist die **Enigma** (siehe die Abbildungen [2.4] und [2.6]). Die Enigma wurde im Jahre 1918 von ARTHUR SCHERBIUS in Berlin entwickelt, zunächst mit dem Ziel der kommerziellen Nutzung der Maschine für die Verschlüsselung von Geschäftskorrespondenz. Diese Maschine wurde in verschiedenen Varianten von der deutschen Wehrmacht während des 2. Weltkriegs als Verschlüsselungsmaschine eingesetzt. Exzellente und umfassende Diskussionen der Enigma findet man in [24,118,120,191] oder [136]. Eine Analyse der bahnbrechenden Tätigkeiten von ALAN TURING zur Dechiffrierung des Enigma–Codes findet man in dem Buch von J. COPELAND [48]. Informative Webseiten zu diesem Thema sind [243] und [244], wo man auch eine Simulation einer Enigma findet.[10]

Im Jahre 1974 veröffentlichte der Kriegsveteran Oberst FREDERICK WILLIAM

[10] Eine beeindruckende Simulation der Wehrmachts–Enigma (und anderer Rotormaschinen) findet man auf der Web Site

http://users.telenet.be/d.rijmenants/en/enigmasim.htm

WINTERBOTHAM ein Buch mit dem Titel *The Ultra Secret* [231], in dem offengelegt wurde, dass es dem britischen Geheimdienst im Zweiten Weltkrieg gelungen war, den Funkverkehr der deutschen Wehrmacht über einen längeren Zeitraum erfolgreich abzuhören. Die in diesem Buch beschriebene Operation *Ultra* war der erfolgreiche Einbruch der britischen Abwehreinheit GC&CS[11] in das Chiffiersystem der deutschen Verschlüsselungsmaschine Enigma.

Abbildung 2.5: Details der Enigma

2.11.1 Aufbau und Arbeitsweise der Enigma

Eine Enigma ist aus folgenden Komponenten aufgebaut (siehe Abbildung [2.5]):

- einer Schreibmaschinentastatur
- einem Steckerbrett (Plugboard)
- drei Rotoren (Walzen)
- einem Reflektor
- einer Anzeigeeinheit bestehend aus Lampen

Die Verschlüsselung mit Hilfe der Enigma durchläuft folgende Schritte (siehe dazu die Abbildung [2.7]):

1. Mit Hilfe der Schreibmaschinentastatur wird der Klartext von einen Operator eingegeben. Das Drücken einer Schreibmaschinentaste hat den Effekt, dass durch einen Kontakt ein elektrischer Stromkreis geschlossen wird.

[11] GC&CS = Government Code and Cipher School, eine Abteilung des britischen Nachrichtendienstes.

Abbildung 2.6: Eine Enigma Rotor Maschine.

❷ Der Strom läuft von der Tastatur durch das Steckerbrett. Zweck dieser Komponente ist es, Paare von Buchstaben durch Kabelverbindungen zu vertauschen. Wird also auf dem Steckerbrett der Buchstabe `A` mit `E` verbunden, wird gleichzeitig dem Buchstaben `E` der Buchstabe `A` zugewiesen. Kryptographisch impliziert dies die Permutation von `A` mit `E` und umgekehrt.

❸ Anschließend läuft der Strom durch die Rotoreinheit mit den Rotoren I, II und III. Ein Rotor ist eine Art Zahnrad mit 26 Kontakten (entspricht den Buchstaben `A - Z`) auf jeder Seite des Rades. Jeder Kontakt auf der einen Seite ist intern mit einem bestimmten Kontakt der anderen Seite fest verdrahtet. Die Rotoren sind mit Gleitkontakten verbunden.

❹ Ein Reflektor — dessen Funktion ist eine weitere Permutation der Buchstaben — spiegelt den Strom und führt diesen an eine andere Position des Rotors III zurück.

❺ Der Strom durchläuft die Rotoren zurück und bringt schließlich eine von 26 Lampen zum Leuchten, die den Chiffretextbuchstaben darstellt.

Die Klartextnachricht wird also wie auf einer normalen Schreibmaschine über die Tastatur eingegeben und der Chiffretext an den Lampen abgelesen. Jedesmal, wenn eine Taste gedrückt wird, ändert der rechte Rotor (*i.e.* Rotor I

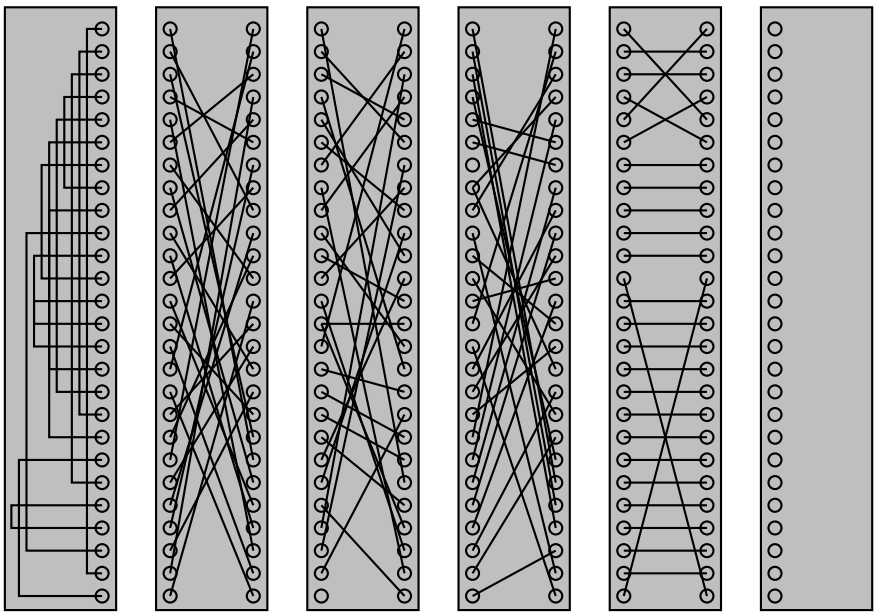


Abbildung 2.7: Das Prinzip der Enigma Rotor Maschine.

in Abbildung [2.7]) seine Einstellung durch eine 1/26 Rotation. Wie bei einem analogen Tachometer ändert sich die Einstellung des mittleren Rotors (Rotor II) um 1/26, wenn der rechte Rotor eine komplette Umdrehung durchgeführt hat. Schließlich dreht sich die Position des Rotors III um 1/26, wenn der mittlere Rotor eine komplette Umdrehung durchgeführt hat. Über kleine Einkerbungen an den Walzen wird die Grundstellung fixiert.

Um eine verschlüsselte Nachricht zu dechiffrieren benötigt man nicht nur eine Enigma Maschine sondern auch die Starteinstellungen, bestehend aus:

1. der Startposition jeder einzelnen Walze.

2. die Lage der Walzen; verschiedene Versionen der Enigma waren mit fünf verschiedenen Walzen ausgerüstet, von denen drei – bei der Marine vier – für die Verschlüsselung benutzt wurden. Diese drei Walzen konnten in beliebiger Reihenfolge in die Maschine eingefügt werden.

3. die Steckerbrett-Verbindungen.

Sind diese Informationen bekannt und wird die Maschine in die gleiche Konfiguration gebracht wie bei der Verschlüsselung, kann der Chiffretext über die Tastatur eingegeben werden und die Lampeneinheit produziert den Klartext.

Abbildung 2.8: Details der Enigma, die Rotoren.

2.11.2 Schlüsselraum der Enigma

Der Schlüsselraum der Enigma ist also gegeben durch die möglichen Startstellungen der Maschine.

1. Jede der drei verwendeten Walzen hat 26 mögliche Startpositionen, dies sind daher

   $$26 \cdot 26 \cdot 26 = 17\,576$$

   mögliche Positionen.

2. Wenn man die Version der fünf Walzen zugrunde legt, hat man

   $$\binom{5}{3} = \frac{5!}{2! \cdot 3!} = 10$$

   unterschiedliche Auswahlmöglichkeiten von drei Walzen. Diese drei gewählten Rotoren können nun in beliebiger Reihenfolge eingebaut werden, dies sind daher $3! = 6$ mögliche Anordnungen.

   Insgesamt ergeben sich also 60 verschiedene Möglichkeiten für die Lage der Walzen.

3. Damit ergibt sich die Größe des Schlüsselraums zu

   $$60 \cdot 17\,576 = 1\,054\,560$$

   verschiedene Rotorstellungen.

4. Eine wesentliche Erhöhung der Sicherheit liefert das Steckerbrett. Für den Einsatz des Steckerbretts gab es auch unterschiedliche Varianten, generell können hiermit maximal 13 Buchstabenpaare vertauscht werden. Wenn nun 5 Paare vertauscht werden, dann führt dies auf

   $$295\,269\,975$$

Abbildung 2.9: Eine Enigma und eine Bombe.

weitere Möglichkeiten der Startkonfiguration der Enigma.

Die Rechnung geht folgendermaßen: Zunächst werden aus 26 Buchstaben zwei ausgewählt, dann aus den verbliebenen 24 Buchstaben zwei Buchstaben, dann aus 22, aus 20 und schließlich aus 18. Die Anzahl der Möglichkeiten, zwei Werte aus k Werten auszuwählen ist durch den Binomialkoeffizienten gegeben. Weiterhin spielt die Reihenfolge keine Rolle, daher ist noch durch 5! zu dividieren. Damit ergeben sich

$$\frac{1}{5!} \cdot \binom{26}{2} \cdot \binom{24}{2} \cdot \binom{22}{2} \cdot \binom{20}{2} \cdot \binom{18}{2} = 295\,269\,975$$

mögliche Buchstabenpaare.

5. Insgesamt hat der Schlüsselraum der Enigma damit

$$295\,269\,975 \cdot 1\,054\,560 = 3\,113\,799\,048\,360\,000$$

verschiedene Werte.

Diese riesige Zahl an Einstellungsmöglichkeiten führte dazu, dass man auf deutscher Seite annahm, dass der Enigma Code nicht zu brechen ist.

Zusammenfassend: Die Sicherheit der Enigma beruht auf den folgenden Grundeinstellungen

- der Rotoranordnung,
- der Rotorstellung,
- der Ringeinstellung
- und den Steckverbindungen.

2.11.3 Einsatz der Enigma

Die Grundeinstellungen der Enigma – i.e. Rotoranordnung, -stellung, Steckverbindungen et. – waren in einem **Codebuch** hinterlegt (siehe Abbildung [2.10]) und wurden täglich geändert. Mit dieser Grundeinstellung konnte nun der gesamte Funkverkehr eines Tages chiffriert werden. Dieses Prozedere hat jedoch den entscheidenden Nachteil, dass Hunderte von Meldungen, die pro Tag verschlüsselt werden mussten, mit ein und dem selben Schlüssel chiffriert wurden. Dies gewährt aber den Kryptoanalytikern eine große Chance, den Schlüssel ausfindig zu machen. Als zusätzliche Vorsichtsmaßnahme wurde deshalb das Verschlüsselungsverfahren dahingehend verfeinert, dass der Tagesschlüssel dazu eingesetzt wurde, für jede Meldung einen einmaligen **Spruchschlüssel** festzulegen.

Bei diesem Verfahren werden die einzelnen Funksprüche zwar mit den im Tagesschlüssel festgelegten Steckerverbindungen und Walzenlagen chiffriert, die *Walzenstellungen* sind jedoch von jedem Operator selbst gewählt. Diese individuellen Walzenstellungen – i.e. der Spruchschlüssel – sind *nicht* Teil des Tagesschlüssels, sind also nicht im Codebuch aufgelistet. Dieser Spruchschlüssel muß daher auf sichere Weise an den Empfänger übermittelt werden, damit dieser den Funkspruch dechiffrieren kann. Dazu wird die Maschine zunächst auf den einheitlichen Tagesschlüssel einjustiert, der natürlich ebenfalls eine bestimmte Walzenstellung festlegt, beispielsweise `QHD`. Im nächsten Schritt wählt der Operator irgendeine Rotorstellung für den Spruchschlüssel, zum Beispiel `DOL`. Dann verschlüsselt er die Zeichenfolge `DOL` mit dem Tagesschlüssel. Um das wasserdicht zu machen wird der Spruchschlüssel gleich zweimal hintereinander in die Enigma eingetippt. Der Sender chiffriert beispielsweise den Spruchschlüssel `DOLDOL` als `KILMRX`. Wesentlich ist, dass die beiden `DOL`-Blöcke unterschiedlich chiffriert sind[12]. Dann werden die Rotoren auf die Positionen `DOL` eingestellt und die eigentliche Nachricht wird damit chiffriert.

Auf Empfängerseite ist die Enigma zunächst auf die Rotorstellung `QHD` eingestellt, dies ist ja der aktuelle Tagesschlüssel. Die ersten sechs Zeichen des empfangenen Chiffrats `KILMRX` werden eingetippt und ergeben `DOLDOL`. Damit ist dem Empfänger bekannt, welchen Tagesschlüssel er wählen muß, justiert die Rotoren neu ein und dechiffriert den Rest der Message.

Enigma-Nachrichten hatten einen Header, der aus

- Absender
- Uhrzeit
- Datum
- Empfängercode

bestand. Potentielle Empfänger von Enigma-verschlüsselten Nachrichten waren Heer, Marine, Luftwaffe etc. Mit Hilfe dieses Empfängercodes war festgelegt, wer der Empfänger der Nachricht war, das heißt, ein Empfänger einer Enigma Nachricht konnte anhand dieses Codes feststellen, ob diese Message überhaupt

[12] Dies wird natürlich dadurch erzielt, dass sich die Walzen nach dem Eintippen jedes Buchstabens um eine Stellung weiterdrehen.

für ihn von Belang war. Der Header wurde im Klartext übertragen. Der Absender wählte anschließend wie oben diskutiert, den Spruchschlüssel und fügte diese chiffrierte, zufällige Startposition zweimal an den Nachrichtenkopf an. Durch diese beiden Buchstabentripletts nach dem Header erhielt der Empfänger nach dem Dechiffrieren die notwendige Information, welche Startposition die drei Walzen haben, um den nachfolgenden Text entschlüsseln zu können. Hinter diesen beiden Tripletts wurden die Nutzdaten angehängt.

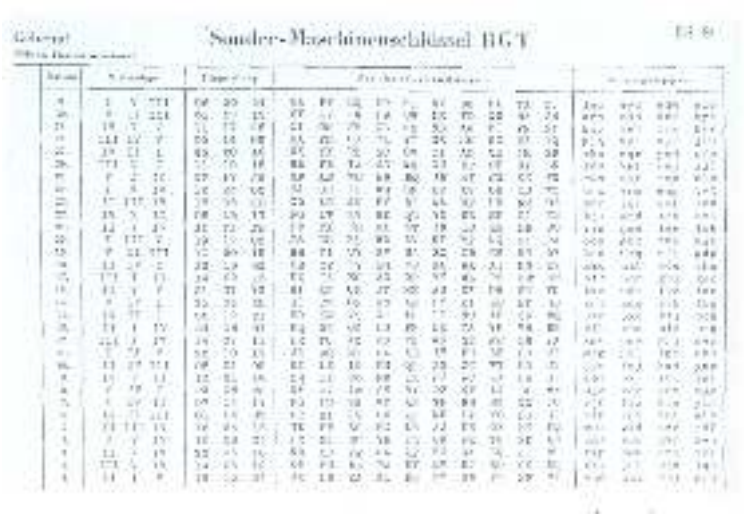


Abbildung 2.10: Monatliche Liste von Anweisungen für die Enigma Verschlüsselung mit Datum, Auswahl und Position von drei Rotoren von fünf möglichen ([110].

Die Länge der verschlüsselten Nachrichten war auf 200 Zeichen beschränkt. Je länger eine Nachricht ist, desto höher ist die Wahrscheinlichkeit, daß sie durch Kryptoanalyse gebrochen werden kann. Falls erfoderlich, wurden daher längere Nachrichten in mehrere Stücke zerteilt. Jede Teilnachricht wurde durch einen anderen Schlüssel chiffriert. Es gab bestimmte Anweisungen und Konventionen, wie Nachrichtenteile, Wörter und Abschnitte zu trennen sind. DiBuchstabenkombination `ch` wurde durch `q` abgekürzt, Worte wurden durch das Zeichen `X` getrennt, Ziffern wurden ausgeschrieben, usw.

2.11.4 Entschlüsselung der Enigma-Chiffre

Die ersten Versuche, den Enigma-Code zu brechen, wurden bereits Mitte der 30er Jahre in Polen durchgeführt. In Polen wurde der Funkverkehr des westlichen Nachbars bereits seit den 20er Jahren abgehört und erfolgreich dechiffriert, um nicht unvorbereitet von Aggressionen des Nachbars überrascht zu werden. Zu Beginn der 30er Jahre war es plötzlich nicht mehr möglich, die Funksprüche zu entschlüsseln, der Grund lag darin, dass die Deutschen die Enigma als Chiffriergerät eingesetzt hatten. Als Folge dessen etablierten die Polen in ihrem

Biuro Szyfrow – die Dechiffriereinheit – einen kleinen Stab von Mathematikern, darunter MARIAN REJEWSKI, deren Aufgabe es war, den Enigma-Code zu brechen. Die Polen waren über französische Geheimdienstkanäle in den Besitz einer Enigma-Maschine und den Bedienungsanweisungen gelangt.

Abbildung 2.11: MARIAN REJEWSKI (1905-1980)

REJEWKSIS Angriffstrategie basierte auf der Tatsache, dass Wiederholungen der Schwachpunkt jedes Verschlüsselungssystems ist. Die offensichtlichen Wiederholungen bei den Engima-verschlüsselten Funksprüchen waren die Spruchschlüssel, die zu Beginn jeder Nachricht gesendet wurden. REJEWSKI bekam täglich eine ganzen Stapel abgefangener Meldungen zur Analyse. Sie alle begannen mit den sechs Buchstaben des wiederholten dreibuchstabigen Spruchschlüssels, alle mit dem gültigen Tagesschlüssel chiffriert. So erhielt er beispielsweise vier Funksprüche, die mit den folgenden Spruchschlüssel begannen.

	1.	2.	3.	4.	5.	6. Buchstabe
1. Funkspruch	L	O	K	R	G	M
2. Funkspruch	M	V	T	X	Z	E
3. Funkspruch	J	K	T	M	P	E
4. Funkspruch	D	V	Y	P	Z	X

Jeweils die ersten und die vierten Buchstaben sind Verschlüsselungen des gleichen Klartextbuchstabens, jeweils die zweiten und fünften sind die Verschlüsselungen des gleichen Klartextzeichens und der dritte und sechste Buchstabe sind ebenfalls Verschlüsselungen des dritten Klartextzeichens im Spruchschlüssel. Die Grundeinstellung der Enigma (i.e. der Tagesschlüssel) (unbekannt) verschlüsselt also den ersten Buchstaben des Spruchschlüssels (unbekannt) mit L und eine spätere Walzenstellung[13] (unbekannt) verschlüsselt den gleichen Buchstaben mit R. Dieser Hinweis ist zwar vage, aber er zeigt, dass die Buchstaben L und R durch die Grundstellung der Enigma in Beziehung stehen. Der zweite Funkspruch in der obigen Liste zeigt, dass M und X korreliert sind,

[13] Also drei Rasterschritte weiter.

der dritte führt auf eine Beziehung zwischen J und M und der vierte verknüpft (irgendwie) die Buchstaben D und P.

Rejewski stellte diese Beziehung in eine Tabelle zusammen:

1. Buchstabe: ABCDEFGHIJKLMNOPQRSTUVWXYZ
2. Buchstabe: P M RX

Wenn nun der Abhördienst genug Funksprüche gesammelt hatte, war Rejewski in der Lage, die Tabelle zu vervollständigen. Beispielsweise ergibt sich folgendes Muster:

1. Buchstabe: ABCDEFGHIJKLMNOPQRSTUVWXYZ
2. Buchstabe: FQHPLWOGBMVRXUYCZITNJEASDK

Der Tagesschlüssel ist unbekannt und ebenfalls der Spruchschlüssel. Diese Tabelle stellt also eine Beziehung der Zeichen des (unbekannten) Spruchsschlüssels dar, der mit dem (unbekannten) Tagesschlüssel chiffriert wurde. Die Frage ist, ob aus dieser Tabelle der Tagesschlüssel abgeleitet werden konnte.

Die Art und Weise, wie Rejewski an diese Problem heranging war, *Ketten* zu konstruieren. Aus der obigen Tabelle sieht man, dass A mit F korreliert ist. Das Zeichen F mit W und W wiederum mit dem Zeichen A. Damit ist die Kette geschlossen. Rejewski untersuchte auch alle anderen Buchstaben im Alphabet nach solchen Beziehungen und stellte die verschiedenen Ketten zusammen:

$$\mathtt{A} \to \mathtt{F} \to \mathtt{W} \to \mathtt{A} \quad \mathbf{3}$$

$$\mathtt{B} \to \mathtt{Q} \to \mathtt{Z} \to \mathtt{K} \to \mathtt{V} \to \mathtt{E} \to \mathtt{L} \to \mathtt{R} \to \mathtt{I} \to \mathtt{B} \quad \mathbf{9}$$

$$\mathtt{C} \to \mathtt{H} \to \mathtt{G} \to \mathtt{O} \to \mathtt{Y} \to \mathtt{D} \to \mathtt{P} \to \mathtt{C} \quad \mathbf{7}$$

$$\mathtt{J} \to \mathtt{M} \to \mathtt{X} \to \mathtt{S} \to \mathtt{T} \to \mathtt{N} \to \mathtt{U} \to \mathtt{J} \quad \mathbf{7}$$

Bisher haben wir nur den ersten Buchstaben des Spruchschlüssels betrachtet. Rejewski analysierte auf die gleiche Weise die Beziehungen zwischen den zweiten und fünften und dritten und sechsten Buchstaben an und stellte sämtliche Ketten und die Zahl ihrer Verknüpfungen auf. Er stellte fest, dass sich diese – neben den Buchstabenfolgen – beiden Parameter jeden Tag änderten. Manchmal ergaben sich viele kurze Ketten, manchmal ein paar längere. Mit anderen Worten, das Muster der Ketten und die Anzahl der Verknüpfungen war eine Folge der jeweiligen Tagsschlüssel. Die verbleibende Frage war, welcher der etwa 10^{15} Schlüssel steckte hinter welchem Muster.

An diesem Punkt kam Rejewski zu einer bahnbrechenden Erkenntnis. Steckerbrett und Walzenkonfiguration zusammen ergeben das exakte Bild der Ketten, jedoch kann der Beitrag der Steckverbindungen, die zur Verschlüsselung dienen, ignoriert werden. Insbesondere hängt die Anzahl der Verbindungen in einer Kette nur von der Walzenstellung ab und nicht von den gewählten Permutationen des Plugboards.

Beispiel:

Angenommen in der obigen Konfiguration ist im Tagesschlüssel die Anweisung vorgesehen, die Buchstaben S und G mittels Steckerverbindung zu vertauschen. Wenn man diesen Bestandteil des Tagesschlüssels ändern, indem man die S-G Verbindung löst und statt dessen T und K vertauscht, dann ändern sich die Ketten folgendermaßen:

A → F → W → A **3**

B → Q → Z → T → V → E → L → R → I → B **9**

C → H → S → O → Y → D → P → C **7**

J → M → X → G → K → N → U → J **7**

Ein paar Buchstaben ändern sich in den Ketten, aber die Gesamtstruktur bleibt erhalten.

Mit anderen Worten, die Anzahl und Länge der Ketten und die Anzahl der Verknüpfungen innerhalb jeder Kette ist eine Funktion der Walzenkonfiguration allein und nicht der Steckverbindungen.

Mit dieser Erkenntnis reduziert sich das Problem von 10^{15} mögliche Schlüsselwerten auf nur noch 105.456 Werte[14]. Um die passende Walzenstellung zu erhalten – und damit den wichtigsten Bestandteil des Tagesschlüssels – ging Rejewski folgendermaßen vor. Da die polnischen Kryptoanalytiker über exakte Nachbauten der Enigma verfügten, machte sich Rejewskis Gruppe an die enorme Aufgabe, jede der 105.456 Walzenkonfigurationen durchzuprüfen und die sich dabei ergebenden Kettenmuster zu erfassen. Nach eine Jahr mühsamer Arbeit verfügte das Kryptoanalytikerteam über einen Katalog, der jeder Walzenstellung das charakteristische Kettenmuster zuordnete.

Obwohl die Gruppe um Rejewski damit den Walzenanteil des Tagesschlüssels gefunden hatte, kannten sie die Steckerverbindungen immer noch nicht. Obwohl es etwa 100 Milliarden Möglichkeiten für diese Verbindungen gibt, ist dieser Teil der Schlüsselrekonstruktion relativ einfach. Zunächst werden die Walzen der Enigma auf die Stellung einjustiert, die dem Tagesschlüssel entspricht. Dann werden alle Steckerverbindungen entfernt und dieses hatte daher keinen Einfluß auf die Chiffrierung. Anschließend wurde eine abgefangene, chiffrierte Nachricht eingetippt. Das ergibt zum größten Teil keinen sinnvollen Klartext, da die Entschlüsselungskomponente Steckerbrett fehlt. Dennoch tauchen dabei ab und zu einigermaßen sinnvolle Wortgebilde auf, zum Beispiel `alkulftilbernil`. Es liegt die Vermutung[15] nahe, dass dieses Fragment für den Klartext *Ankunft in Berlin* steht. Mit dieser Annahme mußten die Buchstaben R und L vertauscht sein, die anderen Buchstaben A, K, U, F, T, I. B und E sind dann nicht vertauscht. Durch die Analyse weiterer Wortfragment ist es dann möglich, sämtliche Buchstabenpaare herauszufinden, die durch eine Steckverbindung vertauscht sind.

[14] Die zu der damaligen Zeit von Rejewski analysierte Enigma hatte nur drei statt fünf Rotoren. Daher reduziert sich die Anzahl der Kombinationen um den Faktor 10 im Vergleich zu der im Abschnit [2.11.2] berechneten Anzahl der Walzenstellungen.

[15] Dies ist ein wesentlicher Bestandteil der Kryptoanalyse, ein hohes Maß an Intuition ist bei dieser Tätigkeit oft sehr hilfreich.

Mit REJEWSKIS Durchbruch bei der Enigma-Entschlüsselung war der deutsche Funkverkehr ein offenes Geheimnis für die polnischen Kryptoanalytiker. Als die Enigma im Laufe der 30er Jahre leicht modifiziert wurde, konnte REJEWSKI durch die Konstruktion einer mechanischen Version seines Katalogsystems die Entschlüsselung der Enigma Funksprüche weiterführen. REJEWSKI mechanisches Katalogsystem funktionierte wie die Enigma selbst und konnte sehr schnell jede der 17.576 Walzenstellungen durchprobieren, bis die richtige gefunden wurde. Aufgrund der sechs möglichen Walzenstellungen arbeiteten sechs dieser Maschinen parallel, jede mit einer der sechs möglichen Walzenanordnungen. Die gesamte Anlage konnte den Tagesschlüssel in etwa zwei Stunden finden. Die mechanischen Katalogsysteme wurden **Bombe** genannt.

Im Jahre 1938 wurde die Enigma erneut verändert, zu den drei Walzen wurden zwei weitere Rotoren hinzugefgt. Dadurch erhöhte sich die Anzahl der möglichen Walzenkonfiguration auf 1 054 560 mögliche Konfiguration, also das Zehnfache an Möglichkeiten. Diese Herausforderung überforderte REJEWSKIS Kryptoanalytiker, da die interne Verdrahtung der beiden zusätzlichen Rotoren unbekannt war und eine Zehnfache Anzahl an Bomben benötigt wurden.

2.11.5 Blechtley Park

Kurz vor Beginn des zweiten Weltkriegs im August 1939 übergaben die polnischen Kryptoanalytiker ihr gesamtes Wissen über die Enigma und deren Entschlüsselung an französische und englische Geheimdienstmitarbeiter. Mit dem Beginn des zweiten Weltkriegs und der Kapitulation Frankreichs im Frühjahr 1940 etablierten die Engländer ein

Abbildung 2.12: Bletchley Park.

2.12 Übungen

1. (Siehe [118, pp. 591–593]) Als im Jahre 1943 das amerikanische Schnellboot PT-109 unter dem Kommando von Lieutenant JOHN F. KENNEDY von einem japanischen Zerstörer versenkt wurde, empfing die australische Küstenwache die folgende Nachricht, die mit der PLAYFAIR Chiffre verschlüsselt war:

```
KXJEY UREBE ZWEHE WRYTU HEYFS
KREHE GOYFI WTTTU OLKSY CAJPO
BOTEI ZONTX BYBNT GONEY CUZWR
GDSON SXBOU YWRHE BAAHY USEDQ
```

   Der verwendete Schlüssel war:

```
royal new zealand navy
```

   Entschlüsseln Sie die Nachricht.

Kapitel 3

Symmetrische Kryptosysteme

3.1 Generelle Eigenschaften von symmetrischen Chiffren

3.1.1 Block Chiffren und Strom Chiffren

Generell unterteilt man Chiffren in **Block Chiffren** und **Strom Chiffren**. Die treffenste Differenzierung zwischen Block– und Stromchiffren ist folgende:[1]

> Block–Chiffren operieren mit festen Transformationen auf großen Blöcken des Klartextes, Strom–Chiffren operieren mit einer zeitabhängigen Transformation auf individuellen Klartextzeichen.

Die Stromchiffrierung ist eine Art der Chiffrierung, bei der die Information entweder zeichenweise oder bitweise kontinuierlich verschlüsselt wird. Die Zeichen eines Datenstroms werden mit einem kontinuierliche *Schlüsselstrom* verknüpft. Üblicherweise werden die Daten– und Schlüsselbits mit der `XOR` Operation verknüpft (siehe Abbildung [3.1]). Dieser Schlüsselstrom hat die gleiche Länge wie der Klartext.

Wie jedes symmetrische Kryptosystem, benutzen Stromchiffren einen Schlüssel K, der auf sicherem Weg zwischen Sender und Empfänger ausgetauscht werden muß. Aus diesem Schlüssel K wird mit Hilfe eines **Pseudozufallszahlengenerators** der **Schlüsselstrom** generiert (siehe auch Abbildung [3.2]). Diesen Prozess führen sowohl Sender als auch Empfänger der Nachricht durch. Der Sender generiert auf diese Weise den Chiffretext. Der Empfänger seinerseits bildet die `XOR` Verknüpfung des Chiffrestroms mit dem Schlüsselstrom. Wegen

$$a \oplus (b \oplus b) = a$$

a = Klartextbit, b = Schlüsselbit kann der Empfänger den Klartext wieder rekonstruieren.

[1] Siehe [183] und [188].

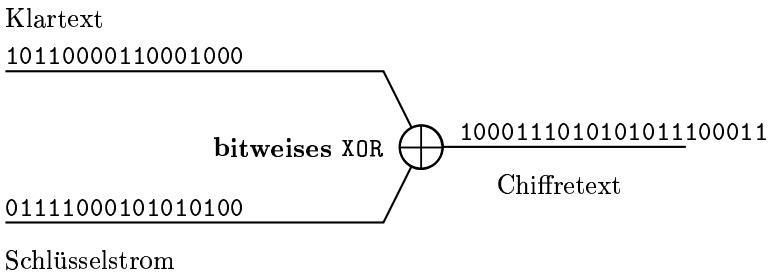


Abbildung 3.1: Prinzip der Stromchiffre.

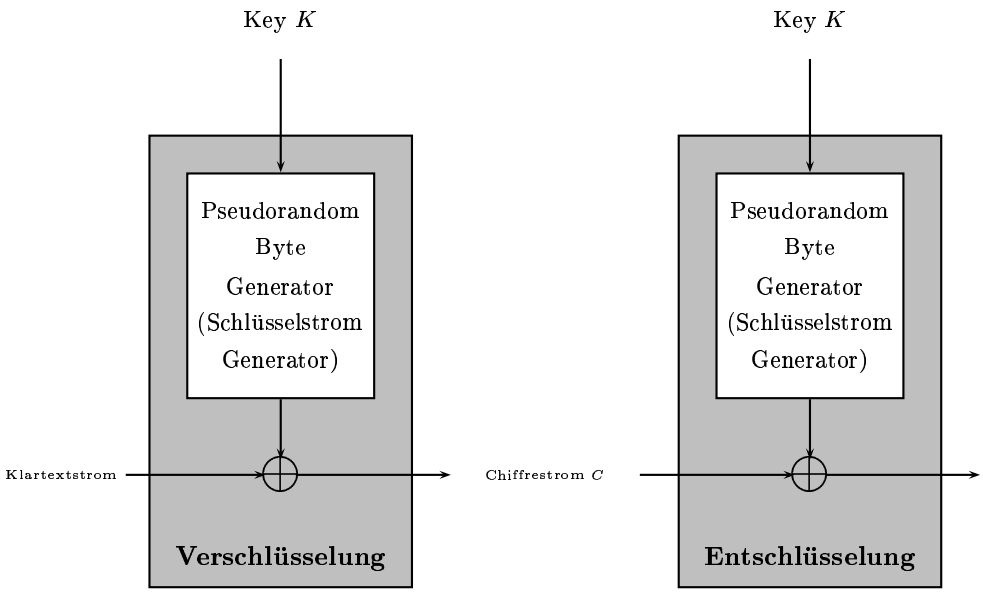


Abbildung 3.2: Prinzipielle Arbeitsweise einer Strom Chiffre

Die Abbildung [3.2] zeigt das Schema einer Stromchiffre. In dieser Struktur läuft ein Schlüssel K als Input in einen Pseudzufallszahlengenerator (siehe Kapitel [19.2]). Dieser erzeugt einen Strom von 8–Bit Zahlen, die (scheinbar) zufällig sind. Ein Strom solcher Pseudozufallszahlen hat die Eigenschaft, dass die Folge der produzierten Zahlen nicht berechenbar ist ohne die Kenntnis des Schlüssels K. Der Output des Generators ist der *Schlüsselstrom*. Dieser wird auf Senderseite byteweise mit dem Klartextstrom über die `XOR` Operation verknüpft. Ist beispielweise der Schlüsselstrom das Byte `01101001` und der Klartextstrom das Byte `10010110`, dann ist der resultierende Chiffretext:

	10010110	Klartext
$\oplus$	01101001	Key Strom
	11111111	Chiffretext

Die Dechiffrierung erfordert den gleichen Schlüsselstrom:

	11111111	Chiffretext
$\oplus$	01101001	Key Strom
	10010110	Klartext

Eine Stromchiffre ist ähnlich wie das One–Time Pad, das in Abschnitt [2.10] diskutiert wurde. Der Unterschied liegt darin, dass das One–Time Pad Verfahren einen echt zufälligen Schlüsselstrom benutzt, während die Stromchiffren eine pseudozufällige Folge von Zahlen als Key Strom verwenden.

3.1.2 Diffusion und Konfusion

Die beiden Konzepte **Diffusion** und **Konfusion** wurden von CLAUDE SHANNON in die Kryptographie eingeführt [199] mit dem Ziel, Kryptosysteme gegenüber statistischen Analysen immun zu machen. Der Hintergrund dabei ist folgender: Angenommen, ein Angreifer hat Kenntnis über statistische Eigenschaften des Klartexts. Beispielsweise sind bei Klartexten in natürlicher Sprache die statistische Häufigkeiten der Buchstaben bekannt. Oder der Klartext enthält sehr wahrscheinlich ganz bestimmte Worte oder Wortgruppen. Falls nun diese statistischen Eigenschaften des Klartexts durch die Chiffriertransformation auf den Chiffretext übertragen wird, kann der Kryptoanalyst mit hoher Wahrscheinlichkeit den Schlüssel, oder Teile des Schlüssels ableiten oder zumindest eine Menge von Schlüsseln eingrenzen, in der der richtige Schlüssel liegt.

SHANNON hat in seiner Arbeit zwei Konzepte herausgearbeitet, mit deren Hilfe Kryptosysteme gegenüber statistischer Kryptoanalyse resistent werden, *Diffusion* und *Konfusion*.[2]

- ✘ **Diffusion** bezeichnet die Eigenschaft einer Chiffre, die statistischen Eigenschaften eines Klartexts so verwischt wird, dass die Zeichen des Chiffretexts keinerlei statistische Häufigkeiten mehr aufweisen. Dies wird in der Regel dadurch realisiert, dass die Änderung eines Klartextzeichens zur Folge hat, dass sich viele Chiffretextzeichen ändern. Weiterhin hat die Änderung eines Zeichens des Schlüssels zur Folge, dass möglichst viele Zeichen des Chiffretextes geändert werden.

  Ziel der Diffusion ist also, die relative Häufigkeiten der Klartextzeichen so zu verwischen, dass die Chiffretextzeichen möglichst gleichverteilt sind. Ähnliches gilt natürlich auch für Bigramme und Trigramme.

  In binären Blockchiffren wird die Diffusion in der Regel durch mehrfache Permutation der Datenbits und anschließender Anwendung einer Funktion auf diese Vertauschung realisiert. Der Effekt ist, dass mehrere Bits aus

[2] Siehe auch [23, Kap. 7.4] oder [113] pp. 118.

verschiedenen Positionen des Klartextblocks zu einem Bit des Chiffretexts beitragen.[3]

- **Konfusion**
  Das Ziel der **Konfusion** ist zu verhindern, dass aus den statistischen Eigenschaften des Chiffretexts Rückschlüsse gezogen werden können auf die statistischen Eigenschaften des Klartextes.

  Die Eigenschaft der Konfusion erhält ein Kryptosystem dadurch, dass komplexe **Substitutionsoperationen** verwendet werden.

3.1.3 Beispiele symmetrischer Chiffren

Es gibt eine Vielzahl symmetrischer Chiffren, hier eine Auswahl.

- Der Data Encryption Standard (DES)
  Der Data Encryption Standard (DES) war von 1977 bis 2002 der von dem NIST zertifizierte Verschlüsselungsstandard. DES ist eine symmetrische Chiffre, eine iterative Blockchiffre mit FEISTEL Struktur.

- Triple DES
  Triple DES ist eine Möglichkeit, die Sicherheit des DES Algorithmus entscheidend zu erhöhen. Triple DES war aus Übergangslösung gedacht, bis ein geeigneter Nachfolger des Data Encryption Standards zertifiziert ist. Beim Triple DES wird ein Klartext drei Mal mit drei verschiedenen Schlüsseln DES chiffriert. Daraus resultiert eine effektive Schlüsselilänge von 168 Bit. Dieses Verfahren, auch 3DES genannt, wurde von vielen Banken für Finanztransaktionen benutzt, es wird auch im Programm `ssh` (Secure Shell) ([7]) benutzt. Eine wesentliche Voraussetzung ist, dass DES keine Gruppenstruktur hat.

- Blowfish
  Blowfish ist eine schnelle, kompakte und einfache symmetrische Blockchiffre, die von BRUCE SCHNEIER entwickelt wurde. Der Algorithmus erlaubt eine variable Schlüssellänge bis 448 Bit und ist optimiert für die Ausführung auf 32–Bit bzw. 64–Bit Architekturen. Der Blowfish Algorithmus ist nicht patentiert und wird in einer Vielzahl von Programmen — darunter auch `ssh` — genutzt.[4]

- IDEA
  Der *International Data Encryption Algorithm* (IDEA) wurde an der ETH Zürich von JAMES L. MASSEY und XUEJIA LAI im Jahre 1990 vorgestellt. IDEA verwendet einen 128 Bit Schlüssel und wird beispielsweise in dem Programm *Pretty Good Privacy* `PGP` eingestzt um Dateien und E–Mails zu

[3] In einigen Kryptographietexten wird Permutation der Diffusion gleichgesetzt. Dies ist in dieser Allgemeinheit inkorrekt wie das Beispiel der Permutationschiffre aus Abschnitt [2.8] zeigt. In dem dort gezeigten Beispiel wurde der Klartext `berufsakademiemannheim` chiffriert. Die statistischen Eigenschaften des entstandenen Chiffretexts sind identisch mit denen des Klartexts (*e.g.* Anzahl der E, M, N etc.).

[4] Siehe die Seite:

`http://www.schneier.com/blowfish.html`

oder [193, pp. 336–339]

verschlüsseln. IDEA ist aufgrund von Patentrechten nicht sehr verbreitet. Siehe [193, pp. 319–325].

✘ RC2
RC2 ist eine von RON RIVEST entwickelte Block Chiffre und von RSA Security als Produkt vertrieben. Der RC2 Algorithmus wurde in einem anonymen Usenet Posting 1996 offengelegt und ist bisher resistent gegen bekannte kryptoanalytische Angriffe. RC2 erlaubt Schlüssel der Länge 1 bis 2048 Bits. Aus US Exportrestriktionen wird das RC2 Kryptosystem üblicherweise mit 40 Bit Schlüssel verwendet.

✘ RC4
Die RC4 Chiffre ist eine ebenfalls von RON RIVEST im Jahre 1987 entwickelte, symmetrische Strom Chiffre, die die Grundlage der Verschlüsselung des WEP Protokolls in WLANs bildet. RC4 wurde ebenfalls als Produkt der Firma RSA Security gehandelt und der Algorithmus wurde aus diesem Grund nicht offengelegt. Wie der RC2 Algorithmus wurde 1994 der RC4 Algorithmus im Usenet anonym publiziert. RC4 ist ebenfalls eine starke Chiffre und erlaubt ebenfalls Schlüssellängen zwischen 1 und 2048 Bits. Die 40 Bit Restriktion der Schlüssellänge für kommerzielle Produkte trifft auf diese Chiffre ebenfalls zu. Details dieser Chiffre findet man in [193, pp. 397-398]. Die Sicherheitsprobleme der WEP Verschlüsselung ist in den Arbeiten [4, 27, 84, 158, 214, 220] offengelegt. Siehe auch [74, Kap. 13.4].

✘ RC5
Die RC5 Chiffre ist eine weitere Block Chiffre von RON RIVEST aus dem Jahre 1995. RC5 erlaubt benutzerdefinierte Schlüssellängen, Größe der zu chiffrierenden Datenblöcke und Anzahl der Verschlüsselungsrunden.

✘ Rijndael
Die Rijndael Block Chiffre wurde von den beiden belgischen Kryptographen JOAN DAEMEN und VINCENT RIJMEN entwickelt und im Jahre 2000 vom NIST als Nachfolger des DES Verfahrens als Advanced Encryption Standard (AES) zertifiziert. Rijndael ist eine sehr schnelle, kompakte Chiffre, die mit den drei Schlüssellängen von 128, 192 und 256 Bit arbeitet (siehe [53, 54, 250]).

✘ Serpent
Serpent ist eine Block–Chiffre von ROSS ANDERSON, ELI BIHAM und LARS KNUDSEN. Serpent hat eine Blockgröße von 128 Bit und variable Schlüssellänge bis 256 Bit. Die Serpent–Chiffre war ein Kandidat für den Advanced Encryption Standard und gehörte mit Rijndael, Twofish, MARS und RC6 zu den Finalisten. Von Experten wurde Serpent als sicherste dieser Kandidaten eingestuft.

✘ CAST–128, CAST–256
Der CAST–Algorithmus ist ursprünglich eine 64–Bit Blockchiffre mit einem 64–Bit Schlüssel, der in den 1990er Jahren von CARLISLE ADAMS und STAFFORD TAVARES entwickelt wurde [193, pp. 334]. CAST ist eine FEISTEL Chiffre. Weitere Entwicklungen erlauben variable Schlüsellängen

von 40 bis 128 Bit, die 256–Bit Schlüsselversion war Kandidat für den Advanced Encryption Standard. [5]

[5] Siehe auch RFC 2411 *The CAST–128 Encryption Algorithm.*

3.2 Die Feistel Chiffre

In der Kryptographie bezeichnet man mit der **Feistel Chiffre** eine Block Chiffre mit einer speziellen Struktur. Diese Chiffre ist benannt nach den IBM Kryptographen Horst Feistel (1915 – 1990), der diese Chiffre in den 70er Jahren bei IBM entwickelte (siehe z.B. [67, pp. 56–59]). Die Feistel Chiffre wird auch als **Feistel Netzwerk** bezeichnet.

Abbildung 3.3: Horst Feistel (1915 – 1990)

Das Feistel Netzwerk bildet die strukturelle Grundlage vieler Block Chiffren, *e.g.*

- Blowfish von Bruce Schneier
- CAST–128
- Data Encryption Standard (DES)
- Fast Data Encipherment Algorithm (FEAL)
- RC5 von Ron Rivest
- Twofish von Bruce Schneier
- Lucifer von Horst Feistel und Don Coppersmith
- MARS
- TEA/XTEA
- MISTY1
- Camellia
- Magenta

Die Feistel Struktur hat den Vorteil, dass Ver– und Entschlüsselung sehr ähnlich, in einigen Implementierungen sogar identisch sind, der Unterschied liegt nur in der Reihenfolge der Anwendung der sogenanten Rundenschlüssel.

3.2.1 Verschlüsselung im Feistel Netzwerk

Die Abbildung [3.4] zeigt die generelle Struktur der von **Horst Feistel** entwickelten Chiffre. Der Input des Verschlüsselungsalgorithmus ist:

- ein Klartextblock der Länge $2w$ Bits

- ein Schlüssel K.

Der Klartextblock wird in zwei Blockhälften L_0 und R_0 mit jeweils w Bits aufgeteilt. Diese beiden Blockhälften laufen durch n **Runden**, werden dabei geeignet verarbeitet, werden schließlich konkateniert und ergeben den Chiffretextblock der Länge $2w$ Bits.

Jede Runde i hat als Input die beiden Blockhälften L_{i-1} und R_{i-1}, die als Output der Verarbeitung der vorhergehenden Runde resultieren, sowie einen **Rundenschlüssel** K_i, der aus dem Key K abgeleitet wird. Alle n Rundenschlüssel sind von K verschieden und sind auch untereinander verschieden.

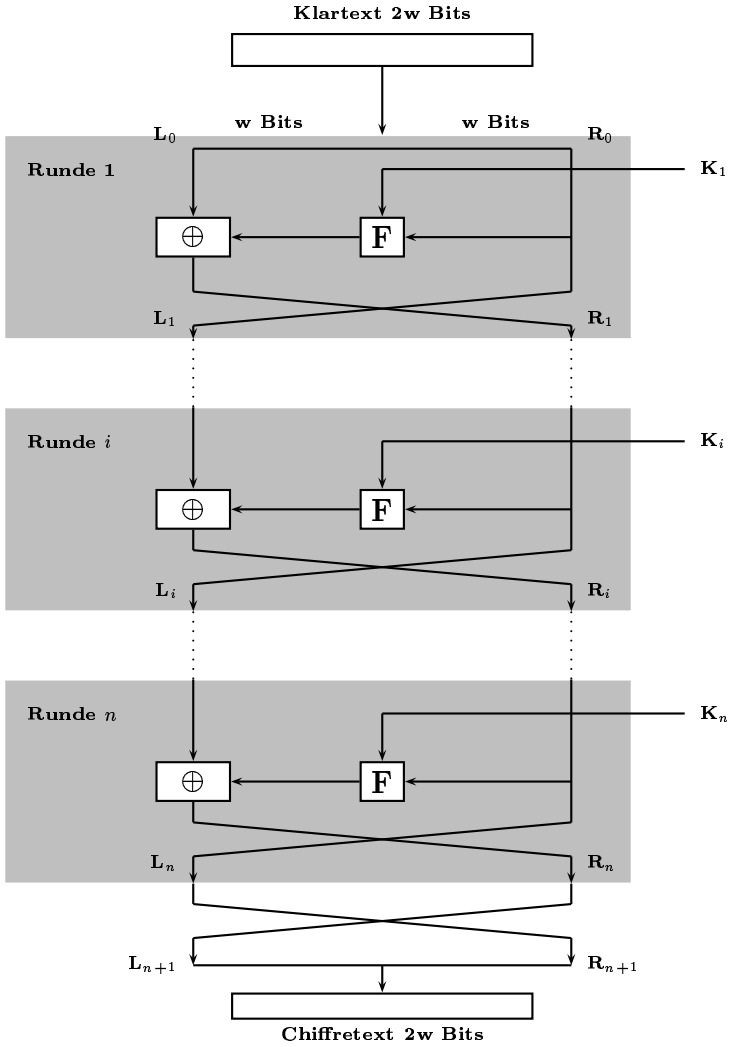


Abbildung 3.4: Das klassische FEISTEL Netzwerk.

Alle Runden haben die gleiche Struktur. In jeder Runde wird eine Substitutionsoperation auf die linke Blockhälfte durchgeführt. Diese Substitutionsoperation erfolgt

- durch die Anwendung einer **Rundenfunktion** auf die rechte Blockhälfte. Diese Rundenfunktion wird mit $\mathbf{F}$ bezeichnet.

- durch die XOR Verknüpfung des Outputs dieser Rundenfunktion mit der linken Blockhälfte.

Die Rundenfunktion $\mathbf{F}$ selbst hat für jede Runde ebenfalls die gleiche Struktur, wird jedoch durch den Rundenschlüssel K_i parametrisiert. Eine Kopie der rechten Blockhälfte läuft unverändert durch die Runde. Nach der Substitu-

tionsoperation findet eine Permutation der beiden Blockhälften statt[6]. Diese Struktur ist eine spezielle Form des Substitutions-Permutations-Netzwerks, das von CLAUDE SHANNON [199] vorgeschlagen wurde.

Die exakte Implementierung eines FEISTEL Netzwerks hängt von den folgenden Parametern ab:

† **Blockgröße:**
Eine größere Blocklänge bedeutet einerseits eine höhere Sicherheit — falls alle Parameter gleich gehalten werden. Jedoch reduziert sich andererseits dadurch die Geschwindigkeit der Ver–/Entschlüsselung. Eine Blockgröße von 64 Bit wird als vernünftiger Kompromiss gesehen und wird in sehr vielen Blockchiffren benutzt.

† **Schlüsselgröße:**
Wie bei der Blockgröße bedeutet eine größere Schlüssellänge eine höhere Sicherheit der Chiffre insbesondere gegen Brute–Force Attacken. Andererseits reduziert eine größere Schlüssellänge die Geschwindigkeit der Ver–/Entschlüsselung. Schlüssel mit eine Länge von 64 Bits und kleiner bieten nach heutigen Standards keine ausreichende Sicherheit mehr. Üblich und empfohlen sind heute symmetrische Kryptosysteme mit einer Schlüssellänge von 128 Bit.

† **Anzahl der Runden:**
Der wesentliche Punkt der FEISTEL Chiffre besteht darin, dass eine einzelne Runde keine adäquate Sicherheit bietet. Erst die Hintereinanderausführung mehrerer Runden lässt die Sicherheit anwachsen. Ein typischer Wert für die Anzahl der Runden ist 16. Auch hier gilt die Faustregel, je mehr Runden desto sicherer ist ein Verfahren, desto aufwändiger ist jedoch die Verarbeitung und desto ineffizienter wird die Verschlüsselung.

† **Algorithmus zur Erzeugung der Rundenschlüssel:**
Eine Implementierung einer FEISTEL Chiffre muß einen Algorithmus beinhalten, der die Rundenschlüssel K_i aus dem Schlüssel K ableitet. Dies muß so geschehen, dass gewährleistet ist:

$$\mathrm{K}_i \neq \mathrm{K}_j \neq K \qquad \forall i, j = 1, \ldots, n, i \neq j$$

Je komplexer das Verfahren der Ableitung der Rundenschlüssel aus den Schlüssel K ist, desto schwieriger ist es für die Kryptoanalyse, die Rundenschlüssel abzuleiten.

† **Rundenfunktion:**
Das Herzstück jeder Implementierung eines FEISTEL Netzwerks ist die Konstruktion der Rundenfunktion F. Je komplexer diese Funktion ist, desto sicherer ist das Kryptosystem gegen kryptoanalytische Angriffe.

[6] Auf die letzte Runde folgt eine zusätzliche Vertauschung der beiden Blockhälften, die die Vertauschung wieder aufhebt, die Teil der letzten Runde ist. Man kann beide Vertauschungen auch einfach aus dem Diagramm weglassen, dies hat aber den unschönen Effekt, dass die Darstellung der Runden etwas an Konsistenz verliert. Der wesentliche Grund, warum in der letzten Runde eine Vertauschung fehlt besteht darin, dass dies die Implementierung des Entschlüsselungsalgorithmus wesentlich erleichtert.

Für die praktische Realisierung einer FEISTEL Chiffre sind zwei weitere Aspekte für das Design der Chiffre wichtig:

- **Schnelle Software Ver–/Entschlüsselung:**
  In sehr vielen Fällen ist der Verschlüsselungsalgorithmus in Anwendungen oder Tools eingebettet.
- **Analyse**
  Obwohl man natürlich daran interessiert ist, den Algorithmus so schwierig wie nur möglich für die Kryptoanalyse zu machen, hat es Vorteile, den Algorithmus so zu designen, dass er für die Analyse leicht zugänglich ist. Falls der Algorithmus leicht und einfach erklärt werden kann, ist es einfacher, das Verhalten dieses Algorithmus unter kryptoanalytischen Angriffen zu analysieren. Dadurch wird ein höherer Grad an Vertrauen in die Stärke des Verfahrens entwickelt. Der Data Encryption Standard hat beispielsweise keine einfach zu analysierende Funktionalität.

3.2.2 Feistel Entschlüsselungsalgorithmus

Das Verfahren der Entschlüsselung in der FEISTEL Chiffre ist im Wesentlichen das gleiche Verfahren wie die Verschlüsselung. Die Vorgehensweise ist folgende: Input des Dechiffrieralgorithmus ist der Chiffretextblock der Länge $2w$ Bit. Der Algorithmus ist exakt der gleiche wie bei der Verschlüsselung. Die Rundenschlüssel K_i müssen in der umgekehrten Reihenfolge angewendet werden. Das heißt, in der ersten Runde wird der Rundenschlüssel K_n, der Rundenschlüssel K_{n-1} in der zweiten Runde usw. und der Key K_1 in der letzten Runde angewendet. Dies ist eine sehr vorteilhafte Eigenschaft einer FEISTEL Chiffre, da ein und der selbe Algorithmus zur Ver- und Entschlüsselung benutzt werden kann.

Um zu sehen, dass der gleiche Algorithmus mit umgekehrter Reihenfolge der Rundenschlüssel das korrekte Ergebnis liefert, betrachten wir die Abbildung [3.5]. Die linke Seite der Abbildung zeigt den Verschlüsselungsalgorithmus, die den Klartext von oben nach unten verschlüsselt. Die rechte Seite ist das Dechiffrierverfahren, das den Chiffretext von unten nach oben in den Klartext zurücktransformiert. Wir verwenden hier die Notation:

- $\mathrm{LE}_i, \mathrm{RE}_i$ bezeichnen Datenblöcke, die durch den **E**ncryption Algorithmus laufen
- $\mathrm{LD}_i, \mathrm{RD}_i$ bezeichnen Datenblöcke, die durch den **D**ecryption Algorithmus laufen.

Die Abbildung deutet an, dass in jeder Runde die Zwischenwerte des Entschlüsselungsverfahrens gleich dem entsprechenden Wert des Verschlüsselungsprozesses ist, wobei die beiden Hälften vertauscht werden. Anders ausgedrückt, ist der Output der i-ten Verschlüsselungsrunde der $2w$ Bit Block

$$\mathrm{LE}_i \,||\, \mathrm{RE}_i,$$

dann ist der entsprechende Input zur $(16-i)$ten Entschlüsselungsrunde der Block

$$\mathrm{RD}_i \,||\, \mathrm{LD}_i.$$

Wir gehen nun die Abbildung [3.5] schrittweise durch, um die Gültigkeit der zugeordneten Blöcke in der Abbildung zu zeigen.[7]

Nach der letzten Iteration des Verschlüsselungsprozesses werden die beiden Output–Hälften vertauscht, so dass der Chiffreblock durch $\mathrm{RE}_{16} \,||\, \mathrm{LE}_{16}$ gegeben ist. Der Output dieser Runde ist der Chiffretext. Wir nehmen nun diesen Chiffreblock als Input in den gleichen Algorithmus. Der Input in die erste Runde ist $\mathrm{RE}_{16} \,||\, \mathrm{LE}_{16}$, dies ist gleich dem 32-Bit Swap der Ausgabe der 16. Runde des Verschlüsselungsprozesses.

Wir zeigen nun, dass die Ausgabe der ersten Runde des Entschlüsselungsprozesses gleich dem 32–Bit Swap des Inputs der 16. Runde des Verschlüsselungsprozesses ist. Dazu betrachten wir zunächst den Verschlüsselungsprozess. Wir sehen, dass gilt:

$$\begin{aligned} \mathrm{LE}_{16} &= \mathrm{RE}_{15} \\ \mathrm{RE}_{16} &= \mathrm{LE}_{15} \oplus \mathrm{F}(\mathrm{RE}_{15}, \mathrm{K}_{16}) \end{aligned}$$

[7] Um das Diagramm zu vereinfachen, sind die Vertauschungen der Blockhälften nach jeder Runde weggelassen. Wichtig ist zu sehen, dass das Zwischenresultat nach Ende der $i$ten Iteration des Verschlüsselungsprozesses durch eine $2w$ Bit Größe gebildet wird, die durch Konkatenation von LD_i und RD_i entsteht.

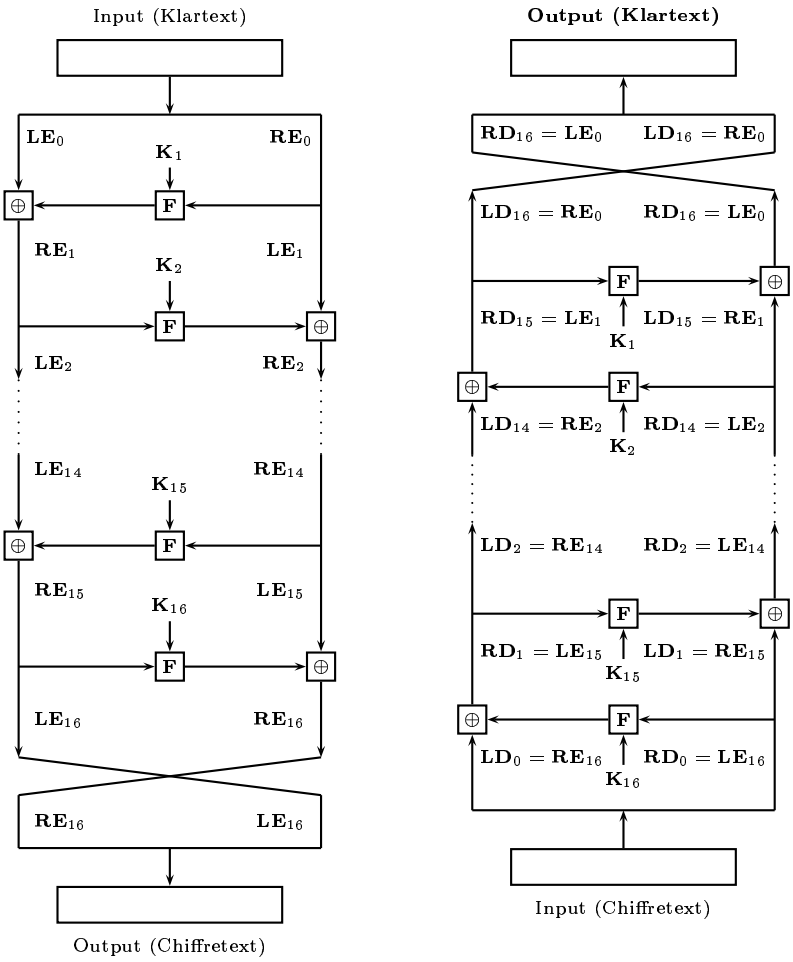


Abbildung 3.5: FEISTEL Ver- und Entschlüsselung.

Auf der Entschlüsselungsseite haben wir:

$$\begin{aligned} \mathrm{LD}_1 &= \mathrm{RD}_0 = \mathrm{LE}_{16} = \mathrm{RE}_{15} \\ \mathrm{RD}_1 &= \mathrm{LD}_0 \oplus \mathrm{F}(\mathrm{RD}_0, \mathrm{K}_{16}) \\ &= \mathrm{RE}_{16} \oplus \mathrm{F}(\mathrm{RE}_{15}, \mathrm{K}_{16}) \\ &= [\mathrm{LE}_{15} \oplus \mathrm{F}(\mathrm{RE}_{15}, \mathrm{K}_{16})] \oplus \mathrm{F}(\mathrm{RE}_{15}, \mathrm{K}_{16}) \end{aligned}$$

Die binäre Operation $\oplus$ — *i.e.* die logische `XOR` Operation — hat die folgende Eigenschaft:

$$\begin{aligned} [A \oplus B] \oplus C &= A \oplus [B \oplus C] \\ A \oplus A &= 0 \\ A \oplus 0 &= A \end{aligned}$$

Aufgrund dessen gilt:

$$\begin{aligned} \mathrm{LD}_1 &= \mathrm{RE}_{15} \\ \mathrm{RD}_1 &= \mathrm{LE}_{15} \end{aligned}$$

Daher ist der Output der ersten Runde des Entschlüsselungsprozesses der 64 Bit Block

$$\mathrm{LE}_{15} \,||\, \mathrm{RE}_{15}$$

Dies ist der 32–Bit Swap des Inputs zur 16. Verschlüsselungsrunde. Diese Korrespondenz setzt sich durch die 16 Iterationen fort. Wir können daher diesen Prozess verallgemeinern. Die Ausgabe der $i$ten Runde des Verschlüsselungsalgorithmus ist

$$\begin{aligned} \mathrm{LE}_i &= \mathrm{RE}_{i-1} \\ \mathrm{RE}_i &= \mathrm{LE}_{i-1} \oplus \mathrm{F}(\mathrm{RE}_{i-1}, \mathrm{K}_j) \end{aligned}$$

Werden nun die Terme umgestellt, wobei die obigen Eigenschaften der XOR Operation verwendet werden, dann erhält man:

$$\begin{aligned} \mathrm{RE}_i &= \mathrm{LE}_{i-1} \\ \mathrm{LE}_i &= \mathrm{RE}_{i-1} \oplus \mathrm{F}(\mathrm{RE}_{i-1}, \mathrm{K}_i) = \mathrm{RE}_{i-1} \oplus \mathrm{F}(\mathrm{LE}_i, \mathrm{K}_i) \end{aligned}$$

Wir haben damit die Eingaben der $i$ten Iteration als Funktion der Ausgaben beschrieben. Diese Gleichungen bestätigen die Zuweisungen auf der rechten Seite der Abbildung [3.5].

3.3 Der Data Encryption Standard

Das bis vor nicht allzu langer Zeit am weitesten verbreitete symmetrische Verschlüsselungsverfahren ist der **Data Encryption Standard** (DES), der im Jahre 1977 vom *National Bureau of Standards* (NBS) — heute **National Institute of Standards and Technology** (NIST)[8]— unter dem *Federal Information Processing Standard* 46 (FIPS PUB 46) (siehe die Spezifikation [56]) veröffentlicht wurde.

Der Data Encryption Standard verschlüsselt einen Klartext in 64 Bit Blöcken. Dabei wird ein 56 Bit Schlüssel benutzt. Der DES Algorithmus transformiert den 64 Bit Input in mehreren Iterationen — insgesamt 16 Runden — in einen 64 Bit Output, den Chiffre Block. Die Entschlüsselung nutzt den gleichen Algorithmus und den gleichen Schlüssel. Darstellungen des Data Encryption Algorithmus findet man beispielsweise in [60,170], [193], Chapter 12, [205], Chapter 3 oder [190, Chap. 5].

In den späten 60er Jahren initiierte IBM ein Forschungsprojekt im Bereich der Computer Kryptographie, geleitet von HORST FEISTEL. Das Projekt wurde 1971 abgeschlossen mit der Entwicklung eines kryptographischen Algorithmus mit der Bezeichnung *Lucifer*.[9] Die LUCIFER Chiffre wurde für LLOYD'S, London entwickelt für den Einsatz in einem Kassensystem. LUCIFER ist eine FEISTEL Block Chiffre, die auf Blöcke der Länge von 64 Bit operiert mit einem Schlüssel der Länge von 128 Bit. Aufgrund der vielversprechenden Resultate des LUCIFER Projektes, begann IBM mit der Entwicklung eines kommerziell einsetzbaren Verschlüsselungsverfahrens, das idealerweise auch auf einem einzigen Chip (als Hardware) implementiert werden konnte. Dieses Entwicklungsprojekt wurde von WALTER TUCHMAN und CARL MEYER geleitet und involvierte nicht nur IBM Mitarbeiter sondern auch externe Berater von der NSA. Das Resultat dieses Projektes war eine verbesserte Version der LUCIFER Chiffre mit verbesserter Resistenz gegenüber bekannten kryptoanalytischen Angriffen, hatte aber eine reduzierte Schlüssellänge von 56 Bit zur Implementierung auf einem einzelnen Chip.

Im Jahre 1973 publizierte das *National Bureau of Standards* (NBS) — diese Behörde firmiert heute unter dem Namen *National Institute of Standards and Technology* (NIST) — eine Ausschreibung für einen nationalen Verschlüsselungsstandard. IBM reichte das Ergebnis des TUCHMAN–MEYER Projekts als Proposal ein. Diese Chiffre wurde schließlich 1977 als **Data Encryption Standard** offiziell verabschiedet.

Bevor die IBM Chiffre als DES Standard verabschiedet wurde, war sie bereits unter schwere Kritik geraten, diese Kritik hält zum Teil bis heute an. Es sind im wesentlichen zwei Kritikpunkte: der 56 Bit Schlüssel und die Rolle der NSA beim Design von DES.

Die Schlüssellänge der ursprünglichen LUCIFER Chiffre ist 128 Bit, die Länge

[8] Siehe auch die Homepage des NIST

www.nist.gov.

[9] In SCHNEIERS Buch [193] wird die LUCIFER Chiffre diskutiert.

der Schlüssel bei der eingereichten Chiffre jedoch nur 56 Bit. Dies ist eine enorme Reduktion von 72 Bits. Kritiker sind der Meinung, dass diese reduzierte Schlüssellänge nicht ausreichenden Schutz gegenüber Brute–Force Attacken bietet.

Mit einer Schlüssellänge von 56 Bit wird ein Schlüsselraum von 2^{56} möglichen Schlüssel zur Verfügung gestellt. Das sind in etwa 7.2×10^{16} Schlüssel. Der reine Zahlenwert scheint eine Brute Force Attack unmöglich zu machen. Nimmt man an, dass im Durchschnitt die Hälfte des Schlüsselraums durchsucht werden muss, bis ein Treffer vorliegt. Betrachtet man eine einzelne Maschine, die pro Mikrosekunde eine DES Entschlüsselung ausführen kann, benötigt diese Maschine mehr als Tausend Jahre, die Chiffre zu brechen. In der Tabelle [3.1] sind eine Reihe von Werten angegeben, aus denen man erkennt, wie die Laufzeit der Schlüsselsuche mit dem Anwachsen der Schlüssellänge wächst.

Schlüsselgr. in Bits	**Größe des alternative keys**	**Zeit mit 1 Entschl./μs**	**Zeit mit 10^6 Entsch./μs**
32	$2^{32} = 4 \cdot 10^9$	35.8 min	2.15 Millisekunden
56	$2^{56} = 7.2 \cdot 10^{16}$	1142 years	10 Stunden
128	$2^{128} = 3.4 \cdot 10^{38}$	$5.4 \cdot 10^{24}$ years	$5.4 \cdot 10^{18}$ Jahre
26 Zeichen	$26! = 4 \cdot 10^{26}$	$6.4 \cdot 10^{12}$ years	$6.4 \cdot 10^6$ Jahre

Tabelle 3.1: Durchschnittliche Zeit für eine Brute Force Attacke.

Schlüsselgr. in Bits	**Schlüssel/ Sekunde**	**Technik**	**Zeit mit 10^6 Entsch./μs**
40	10	10 Jahre alter Desktop	3.484 Jahre
40	1.000	Heutiger Desktop	35 Jahre
40	10^6	Kleines Netz	13 Tage
40	10^9	Mittelgroßes Netzwerk	18 Minuten
56	10^6	Zukünftiger Desktop	2.283 Jahre
56	10^9	Mittelgroßes Netzwerk	2,3 Jahre
56	100×10^9	DES Cracker	8 Tage
64	10^9	Mittelgroßes Netzwerk	585 Jahre
80	10^6	Kleines Netz	38×10^9 Jahre
80	10^9	Mittelgroßes Netzwerk	38×10^6 Jahre
128	10^9	Mittelgroßes Netzwerk	10^{22} Jahre
128	10^{18}	GRID Computing	10^{10} Jahre
128	10^{23}	Quanten Computer (?)	108×10^6 Jahre
192	10^9	Mittelgroßes Netzwerk	2×10^{41} Jahre
192	10^{18}	GRID Computing	2×10^{32} Jahre
192	10^{23}	Quanten Computer (?)	2×10^{27} Jahre
256	10^{23}	Quanten Computer (?)	$3,7 \times 10^{46}$ Jahre

Tabelle 3.2: Detailliertere Darstellung von Brute Force Attacken ([92]).

Die Annahme einer Entschlüsselung pro Mikrosekunde ist jedoch sehr konservativ. Bereits 1977 postulierten DIFFIE und HELLMAN die Möglichkeit ([65]), mit der damals bestehenden Technologie den Bau einer Parallelmaschine mit 1 Million Verschlüsselungseinheiten. Jede dieser Einheiten ist in der Lage, eine Ver−/Entschlüsselung pro Mikrosekunde durchzuführen. Solch eine Maschine reduziert die durchschnittliche Suchzeit nach dem passenden Schlüssel auf 10 Stunden. Die Autoren bestimmten die Kosten – nach dem damaligen Niveau – solch einer Maschine auf etwa 20 Millionen US$. Dies ist nicht viel für eine Behörde wie die NSA.

Der zweite Kritikpunkt an der Entwicklung des Data Encryption Standards richtete sich auf die Design Kriterien der inneren Struktur von DES, insbesondere einen Teil der Rundenfunktion **F**, die sogenannten **S-Boxen**. Das genaue Design dieser Elemente wurde von der NSA entwickelt und unterlagt (und tut's immer noch) der Geheimhaltung. Als Folge dessen war man sich als Benutzer der DES Algorithmus nie sicher, ob die intere Struktur des DES frei von irgendwelchen versteckten Schwachpunkten war, die es der NSA ermöglichen, eine mit dem DES verschlüsselte Nachricht ohne Kenntnis des Schlüssels zu lesen.

Abbildung 3.6: Das Hauptquartier der NSA in Baltimore, USA (aus [191]).

Nachfolgende Analysen des DES durch Kryptographen haben jedoch keinerlei Hintertüren offenbart, im Gegenteil, das Design der Rundenfunktion **F** von DES zeigt erstaunliche Robustheit und Resistenz gegenüber vielen Angriffsversuchen (e.g. Differentielle Kryptoanalyse).

Einige Meilensteine in der Geschichte des Data Encryption Standards sind:

vor 1970 Kryptographie wurde hauptsächlich für militärische und diplomatische Zwecke eingesetzt. Aufgrund der Entwicklung, des Aufkommens und der Verbreitung der digitalen Informationsverarbeitung wuchs der Bedarf an Datenverschlüsselungsverfahren und Methoden zur Authentifikation.

Mai 1973 Das NIST (National Institute of Standards and Technology, USA)

publiziert eine öffentliche Ausschreibung für einen kryptographische Algorithmus Standard.

August 1974 Aufgrund fehlender qualifizierter Vorschläge startet das NIST eine zweite Ausschreibungsrunde. Schließlich wurde von IBM ein geeigneter Algorithmus eingereicht.

August 1974 Details eines symmetrischen Verschlüsselungs Verfahrens, entwickelt durch die IBM und getestet durch die NSA, werden veröffentlicht.

15.6.77 Veröffentlichung der endgültigen Version des Data Encryption Standards. Die Standard Chiffre wird alle fünf Jahre auf ihre Sicherheit hin überprüft.

1981 Das ANSI Gremium (American National Standards Institute) erkennt den Data Encryption Standard als Standard-Verschlüsselungsverfahren an.

1982-1986 Basierend auf dem DES Algorithmus werden neue Standards für PIN Verschlüsselung, Verschlüsselung in der Telekommunikation sowie der FAX Übertragung freigegeben.

1983 Review des DES Algorithmus ohne eine Beanstandung.

1988 Nach heftigen Diskussionen wird der DES Algorithmus erneut zertifiziert (FIPS 46–1).

1993 Erneute Zertifizierung von DES in FIPS 46–2; es ist jedoch bereits absehbar, dass in der kommenden Dekade der DES Algorithmus nicht mehr die notwendige Sicherheit bietet.

1994 Erfolgreiche Attacke auf den DES. In 49 Tagen wird eine mit dem DES verschlüsselte Nachricht von 12 HP-9735 Workstations dechiffriert.

1997 Das NIST kündigt die Entwicklung des Advanced Encryption Standards (AES) an.

17.7.98 DES wird innerhalb von drei Tagen geknackt.

19.1.1999 Der Electronic Frontier Foundation gelingt es, mit Hilfe eines speziell ausgerüsteten Supercomputers (DES Cracker, siehe Abbildung [3.7]) und 100.000 Rechnern den DES innerhalb von 22 Stunden und 15 Minuten zu brechen.[10]

Nov. 2001 Der Advanced Encryption Standard (AES) — *i.e.* der RIJNDAEL Algorithmus — löst den DES als offiziell zertifizierte Standard Chiffre ab (FIPS 197).

19. Mai 2005 Das NIST verkündet öffentlich, dass der DES nicht mehr der Standard Algorithmus ist.

[10] Siehe die EFF Page

http://www.eff.org/Privacy/Crypto/

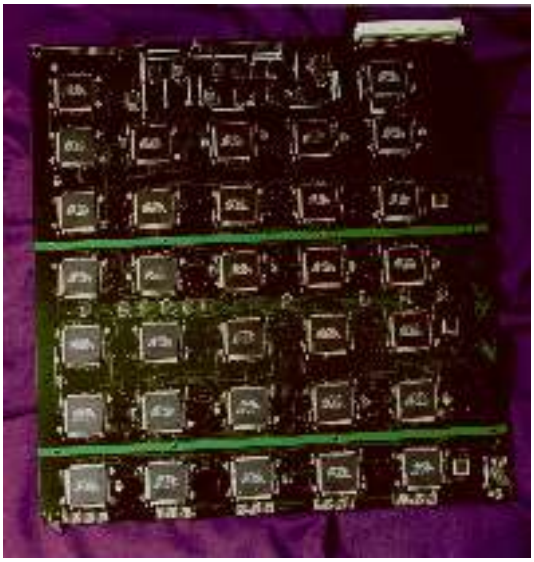


Abbildung 3.7: Mainboard der DES-Cracking Machine der Electronic Frontier Foundation (EFF) (siehe [255]).

3.3.1 Ein Demo–Modell

In diesem Abschnitt sehen wir uns ein vereinfachtes Modell des Data Encryption Standards an, das die wesentlichen Eigenschaften des realen Verschlüsselungsverfahrens beinhaltet, aber von der Schlüssel– und Blockgröße soweit reduziert ist, dass die einzelnen Schritte des Algorithmus noch nachvollziehbar sind.[11]

Die Abbildung [3.8] illustriert die generelle Struktur des Spiel-Modells, das wir kurz mit S-DES bezeichnen. Die S–DES Verschlüsselung hat als Input einen 8–Bit Datenblock als Klartext — *e.g.* `1011 1101` — und einen 10 Bit Schlüssel. Der Output ist ein 8–Bit Block, der den Chiffretextblock darstellt.

Der S–DES Entschlüsselungsalgorithmus arbeitet genau umgekehrt: Input ist ein 8-Bit Chiffretextblock und der gleiche 10 Bit Schlüssel. Der Output ist der ursprüngliche 8–Bit Klartextbock.

Wie aus der Abbildung [3.8] abzuleiten ist, beinhaltet der Verschlüsselungsalgorithmus fünf Funktionen:

- Eine Startpermutation der Klartextbits (initial permutation) (**IP**),
- eine komplexe Funktion, die in der Abbildung [3.8] mit f_K bezeichnet ist. Diese setzt sich sowohl aus die sowohl Permutations– als auch aus Substitutionsoperationen zusammen und hängt von einem Rundenschlüssel K_1 ab,
- eine einfache Vertauschung (swap) (**SW**), die die beiden Blockhälften vertauscht

[11] Die folgende Darstellung des DES als Spielmodell findet man in STALLINGS' Buch [205].

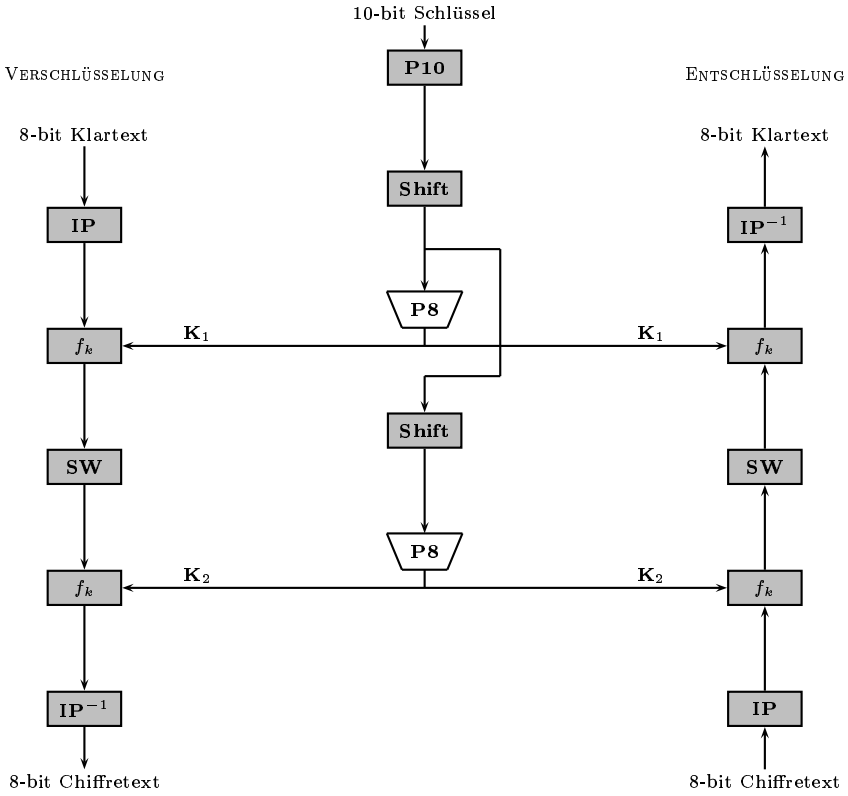


Abbildung 3.8: Das generelle Schema des Demo-DES Algorithmus.

- ein zweites Mal die Funktion f_{K}, diesemal mit einem anderen Rundenschlüssel K_2 parametrisiert.
- und eine zweite Permutation, die die Inverse der ursprünglichen Permutation ist ($\mathbf{IP}^{-1}$).

Die Verwendung von Permutationen und Substitutionen in mehreren Stufen resultiert in einem komplexeren Algorithmus, was zur Folge hat, dass die Kryptoanalyse schwieriger wird.

Wie in der Abbildung [3.8] angedeutet ist, hat die Funktion f_{K} nicht nur den Datenblock als Input sondern auch einen 8-Bit Rundenschlüssel, der mit K_1 bezeichnet ist. Dieser 8-Bit Rundenschlüssel wird aus dem 10-Bit Schlüssel abgeleitet, wie der mittlere Verarbeitungsprozess der Abbildung [3.8] zeigt. Der 10 Bit Key läuft zunächst durch eine Permutation **P10**. Anschließend erfolgt eine sogenannte Shift Operation. Der Output dieser Operation läuft in die Operation **P8**, dies ist eine Permutation und Projektion der 10 auf 8 Bit. Die Ausgabe der Shift Operation wird in eine weitere Instanz der **P8** Operation geleitet, dies produziert den zweiten Rundenschlüssel K_2.

Damit kann der Verschlüsselungsalgorithmus folgendermaßen formalisiert werden

$$\mathrm{IP}^{-1} \circ f_{\mathrm{K}_2} \circ \mathrm{SW} \circ f_{\mathrm{K}_1} \circ \mathrm{IP}.$$

Dies können wir auch folgendermaßen schreiben:

$$\text{Chiffretext} = \mathrm{IP}^{-1}(f_{\mathrm{K}_2}(\mathrm{SW}(f_{\mathrm{K}_1}(\mathrm{IP}(\text{Klartext}))))),$$

mit den Rundenschlüsseln:

$$\mathrm{K}_1 = \mathrm{P8}(\mathrm{Shift}(\mathrm{P10}(\text{key})))$$

und

$$\mathrm{K}_2 = \mathrm{P8}(\mathrm{Shift}(\mathrm{Shift}(\mathrm{P10}(\text{key})))).$$

Die Entschlüsselung ist in der Abbildung [3.8] ebenfalls skizziert, im wesentlichen ist es der Umkehrprozess der Verschlüsselung:

$$\text{Klartext} = \mathrm{IP}^{-1}(f_{\mathrm{K}_1}(\mathrm{SW}(f_{\mathrm{K}_2}(\mathrm{IP}(\text{Chiffretext}))))).$$

Wir betrachten nun die verschiedenen Teile des Algorithmus im Detail.

Schlüsselerzeugung

Das S–DES Verfahren benutzt einen 10–Bit Schlüssel, der zwischen Sender und Empfänger ausgetauscht werden muss, daher zählt S–DES — und natürlich auch DES — zu der Klasse der symmetrischen Kryptosysteme. Aus diesem 10–Bit Schlüssel werden zwei 8–Bit Rundenschlüssel abgeleitet, die in bestimmten Zwischenstufen des Ver– und Entschlüsselungsalgorithmus eingehen. Die Abbildung [3.9] zeigt die Details der Schlüsselerzeugung.

① Wir beginnen mit einen 10 Bit Schlüssel, den wir mit

$$(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$$

bezeichnen. Die Permutation **P10** ist definiert durch:

$$\mathrm{P10}(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_1, k_9, k_8, k_6)$$

Das bedeutet, das Inputbit 1 (Position k_1) wird zum Outputbit 7, Inputbit 2 zum Outputbit 3 usw. Als Beispiel betrachten wir den 10 Bit Key

$$\mathrm{K} = \texttt{10100 00010}$$

Dieser Schlüssel wird abgebildet auf:

$$\begin{aligned}\mathrm{P10}(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) &= \mathrm{P10}(\texttt{1,0,1,0,0,0,0,0,1,0}) \\ &= (\texttt{10000 01100})\end{aligned}$$

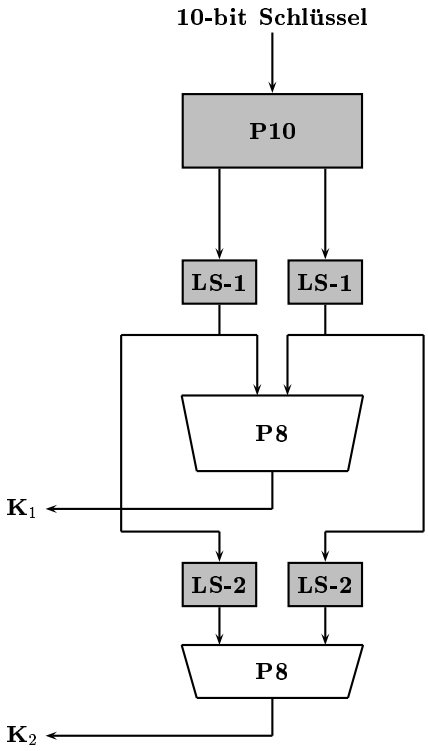


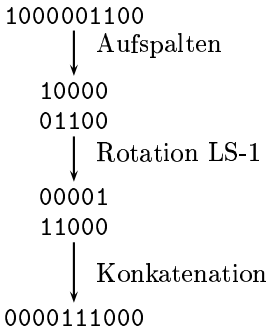
Abbildung 3.9: Algorithmus zur Erzeugung der Rundenschlüssel im S-DES.

② Der nächste Schritt in der Erzeugung der Rundenschlüssel besteht in der Durchführung einer Left Shift Operation. Das Prozedere funktioniert explizit folgendermaßen.

(a) Der Output der **P10** Operation wird in zwei 5 Bit Blöcke aufgespaltet.

(b) Anschließend wird auf jeden der beiden 5–Bit Blöcke eine Left–Shift Operation um eine Position angewendet, wobei das Bit ganz links wieder rechts angefügt wird (Rotation)

Für unser Beispiel sieht dies folgendermaßen aus:



③ Der nächste Schritt besteht in der Anwendung der Operation **P8** auf den Output der Shiftoperation, den wir mit $(k'_1, k'_2, \ldots, k'_{10})$ bezeichnen. Die Operation **P8** wählt acht aus zehn Bits aus und führt eine Permutation durch:

$$\mathrm{P8}(k'_1, k'_2, k'_3, k'_4, k'_5, k'_6, k'_7, k'_8, k'_9, k'_{10}) = (k'_6, k'_3, k'_7, k'_4, k'_8, k'_5, k'_{10}, k'_9)$$

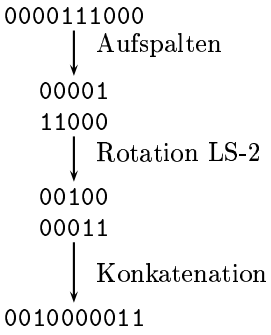
Das Resultat ist der Rundenschlüssel K_1.

Für unser Beispiel resultiert der Rundenschlüssel:

$$\begin{aligned}\mathrm{P8}(k'_1, k'_2, k'_3, k'_4, k'_5, k'_6, k'_7, k'_8, k'_9, k'_{10}) &= \mathrm{P8}(0,0,0,0,1,1,1,0,0,0)\\ &= (1,0,1,0,0,1,0,0)\\ &= \mathrm{K}_1\end{aligned}$$

④ Um den zweiten Rundenschlüssel zu erhalten, geht man zurück zu dem Block, der aus der Anwendung der beiden LS-1 Operationen auf die 5–Bit Teilblöcke resultiert. Dieser 10 Bit Block wird erneut in zwei 5–Bit Teilblöcke aufgespaltet und jeder Teilblock wird erneut rotiert, diesmal aber um zwei Positionen.

Für unser Beispiel ergibt dies:



⑤ Anschließend wird erneut die **P8** Operation angewendet um den Rundenschlüssel K_2 zu erhalten. Das Ergebnis unseres Beispiels ist

$$K_2 = (01000011)$$

S–DES Verschlüsselung

Wie bereits erwähnt, beinhaltet die S–DES Verschlüsselung die sequentielle Anwendung von fünf Funktionen. Diese Schritte sind im Detail in der Abbildung [3.10] dargestellt.

① **Initial Permutation**
Der Input ist ein 8–Bit Klartextblock,

$$(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$$

der im ersten Schritt durch eine Permutationsoperation (die sog. *Initial Permutation*) **IP** läuft. Diese Vertauschung ist folgendermaßen definiert:

$$\mathrm{IP}(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8) = (p_2, p_6, p_3, p_1, p_4, p_8, p_5, p_7)$$

Diese Permutation erhält alle 8 Bits des Klartexts, mischt sie aber durch.

② **Die Rundenfunktion f_K**
Die komplexeste Komponente des S–DES Algorithmus ist die Rundenfunktion f_K, die aus einer Kombination von Permutations- und Substitutionsoperationen besteht. Diese Operationen können folgendermaßen dargestellt werden. Seien L und R die beiden 4-Bit Blockhälften des 8-Bit Blocks, der in die Rundenfunktion f_K läuft. Die dabei auftretende Funktion F, die von einem Rundenschlüssel SK (für *subkey*) abhängt, ist eine Abbildung, die 4–Bit Strings in 4–Bit Strings abbildet:

$$F : \mathbb{Z}_2^4 \longrightarrow \mathbb{Z}_2^4$$

Damit können wir die Rundenfunktion f_K darstellen als

$$f_K(L, R) = (L \oplus F(R, SK), R)$$

$\oplus$ ist die bitweise exclusive OR Funktion `XOR`.

Beispiel:

Angenommen der Output der IP Operation in der Abbildung [3.10] ist der 8–Bit String (10111101) und F(1101,SK) = (1110) für irgendeinen Rundenschlüssel SK. Dann ist

$$\begin{aligned} f_K(10111101) &= (1011 \oplus F(1101, SK), 1101) \\ &= (1011 \oplus 1110, 1101) \\ &= (01011101) \end{aligned}$$

③ Die Abbildung F — Teil der Rundenfunktion f_{K} — hat als Input vier Bits, die wir mit (n_1, n_2, n_3, n_4) bezeichnen. Die erste Operation von F ist eine Expansions/Permutationsoperation, die in der Abbildung [3.10] mit **E/P** bezeichnet ist. Diese **E/P** Operation produziert aus einem 4–Bit Input einen 8–Bit String als Output. Im Detail:

$$\mathbf{E/P}(n_1 n_2 n_3 n_4) = (n_4 n_1 n_2 n_3 n_2 n_3 n_4 n_1)$$

Für die folgenden Schritte ist es zweckmäßig, diese acht Bit in zwei Viererblöcke zu trennen und die folgende Notation zu verwenden:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

Der 8–Bit Rundenschlüssel K_1 hat die Komponenten:

$$K_1 = \big(k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}, k_{17}, k_{18}\big)$$

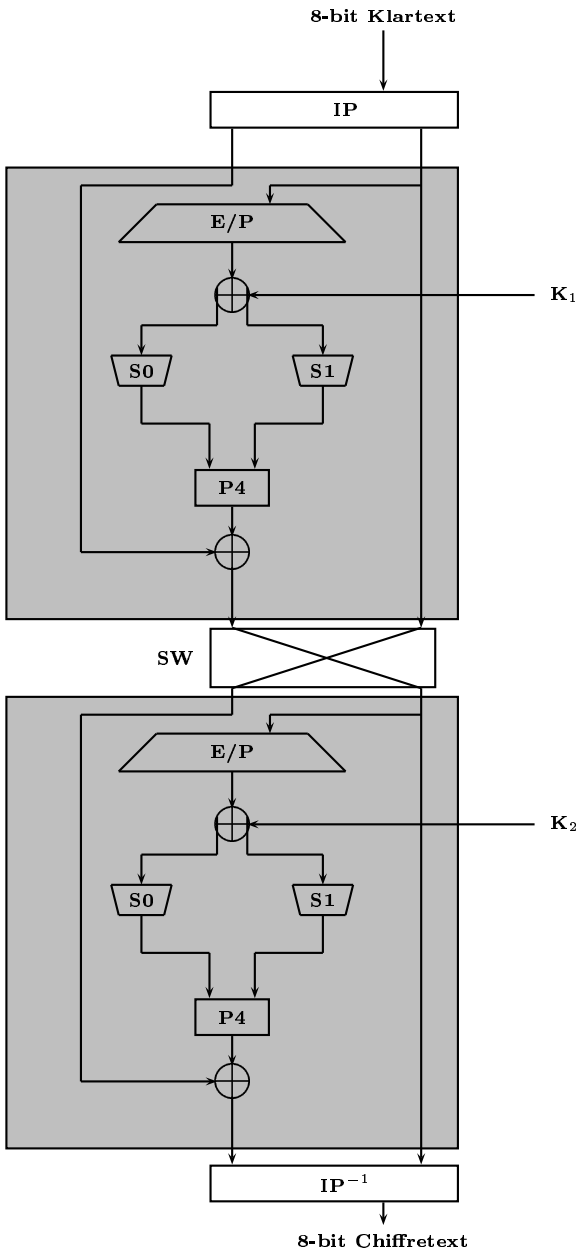


Abbildung 3.10: Details des S-DES Verschlüsselungsverfahrens.

Dieser wird mit dem Output der **E/P** Operation auf folgende Weise `XOR` verknüpft:

$$\begin{array}{c|c|c|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Um den Sachverhalt übersichtlich zu halten und die Schreibarbeit zu erleichtern, werden diese 8 Bits folgendermaßen umbenannt: Wir führen

zwei 4–Bit Blöcke p_0 und p_1 ein, die aus den Bits

$$p_0 = p_{0,0}, p_{0,1}, p_{0,2}, p_{0,3}$$
$$p_1 = p_{1,0}, p_{1,1}, p_{1,2}, p_{1,3}$$

aufgebaut sind. Diese Bits definieren wir über:

$$\begin{array}{l|lll|l} p_{0,0} = n_4 \oplus k_{11} & p_{0,1} = n_1 \oplus k_{12} & p_{0,2} = n_2 \oplus k_{13} & p_{0,3} = n_3 \oplus k_{14} \\ p_{1,0} = n_2 \oplus k_{15} & p_{1,1} = n_3 \oplus k_{16} & p_{1,2} = n_4 \oplus k_{17} & p_{1,3} = n_1 \oplus k_{18} \end{array}$$

Die ersten vier Bits

$$p_0 = p_{0,0}, p_{0,1}, p_{0,2}, p_{0,3}$$

laufen in die **S–Box S0**; die S-Box bildet diese vier Input Bits auf zwei Output Bits ab.

Auf die gleiche Weise werden die restlichen vier Bits

$$p_1 = p_{1,0}, p_{1,1}, p_{1,2}, p_{1,3}$$

durch die zweite S–Box **S1** auf zwei andere Bits abgebildet.

Die beiden S–Boxen sind folgendermaßen definiert:

$$\mathbf{S0} = \begin{pmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{pmatrix}$$

und:

$$\mathbf{S1} = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{pmatrix}$$

Die S–Boxen — diese Elemente führen eine *Substitution* durch — operieren auf folgende Weise: Das erste und vierte Input–Bit werden als 2–Bit Zahl interpretiert und definieren eine Zeile der S–Box. Das zweite und das dritte Bit definieren als binäre Zahl interpretiert eine Spalte der S–Box. Der binäre Wert des auf diese Weise spezifizierten Matrixelementes ist der 2–Bit Output der Box.

Beispiel:

Der Input der **S0** Box ist der 4-Bit String

$$\begin{aligned} p_0 &= (p_{0,0}, p_{0,1}, p_{0,2}, p_{0,3}) \\ &= (0, 1, 0, 0) \end{aligned}$$

Das erste und das letzte Bit, i.e. die beiden Bits

$$(p_{0,0}, p_{0,3}) = (0, 0)$$

definieren die Zeile 0 der **S0** Box, da 00 als Binärzahl den dezimalen Wert 0 ergibt — damit die erste Zeile der Box. Die mittleren beiden Bits:

$$(p_{0,1}, p_{0,2}) = (1, 0)$$

ergeben den dezimalen Wert 2, damit ist die dritte Spalte der **S0** Box festgelegt. Das derart spezifierte Matrixelement — also Zeile 1, Spalte 3[12] — hat den dezimalen Eintrag 3, also binär die beiden Bits 11, was den Output dieser Operation darstellt.

Auf die gleiche Weise werden die Bits $(p_{1,0}, p_{1,3})$ und $(p_{1,1}, p_{1,2})$ verwendet, um ein Matrixelement der **S1** Box zu indizieren, was zwei weiteren Bits produziert.

Wir bezeichnen den Output der beiden S–Boxen mit

$$(s0_1, s0_2, s1_1, s1_2).$$

Diese vier Bits werden durch die Permutationsoperation **P4** wie folgt nochmals permutiert:

$$\mathbf{P4}(s0_1, s0_2, s1_1, s1_2) = (s0_2, s1_2, s1_1, s0_1)$$

Der Output der **P4** Operation ist gleichzeitig der Output der **F** Funktion.

④ **Die Switch Funktion SW**
Die Rundenfunktion f_K ändert lediglich die vier linken Bits des 8–Bit Klartextblocks. Die Switch Funktion **SW** vertauscht die linke und rechte Blockhälfte, so dass in der zweiten Runde die Rundenfunktion f_K auf den anderen vier Bits operiert. In dieser zweiten Runde sind die Operationen **E/P, S0, S1** und **P4** die gleichen wie in der ersten Runde, der Rundenschlüssel ist nun aber durch K_2 gegeben.

⑤ **Permutation**
Der letzte Schritt des S-DES Algorithmus ist eine 8–Bit Block Permutation, die die inverse Transformation zu **IP** darstellt. Diese inverse Permutation ist:

$$\mathrm{IP}^{-1}(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8) = (p_4, p_1, p_3, p_5, p_7, p_2, p_8, p_6)$$

mit

$$\mathrm{IP}^{-1}(\mathrm{IP}(X)) = X$$

[12] Die Zählung beginnt mit 0.

3.3.2 Reales DES

In diesem Abschnitt sehen wir uns die Details des DES Algorithmus in der NIST Spezifikation an.

DES Verschlüsselung

Die Grobstruktur des DES Verschlüsselungsverfahrens ist in der Abbildung [3.11] dargestellt. Wie jedes Kryptosystem gibt es zwei Inputs für den Verschlüsselungsalgorithmus:

1. den zu verschlüsselnden Klartext
2. den Schlüssel K.

Der Data Encryption Standard arbeitet mit einer Klartextblocklänge von 64 Bit und einer effektiven Schlüssellänge von 56 Bit.

Die linke Seite der Abbildung zeigt, dass die Verarbeitung des Klartextblocks in drei Phasen durchgeführt wird:

- ☞ Zunächst läuft der Klartextblock durch eine Start-Permutation (*initial permutation*) **IP**, welche die Bits in fest vorgegebener Weise – analog der **IP**-Operation des S-DES Verfahrens – umordnet.
- ☞ Diesem ersten Schritt folgt eine Phase, die aus 16 Verarbeitungsrunden besteht. In jeder Runde werden Permutations- und Substitutionsoperationen angewendet
- ☞ Die linke und rechte Hälften des Outputs der 16 Runden werden vertauscht und liefern den sogenannten **Präoutput**.
- ☞ Zuletzt wird eine Permutation ausgeführt, diese ist die inverse Permutation der Start-Permutation. Der Output dieser Transformation ist der 64-Bit Chiffretextblock.

Die rechte Seite der Abbildung [3.11] zeigt die Ableitung der Rundenschlüssel aus dem DES-Schlüssel K. Zunächst läuft der Schlüssel durch eine Permutationsoperation. Dann wird für jede der 16 Runden ein Rundenschlüssel K_i erzeugt. Dies geschieht durch eine Kombination von Left-Shift und Permutationsoperationen. Die Permutationsoperation ist die gleiche für jede Runde, unterschiedliche Rundenschlüssel entstehen durch unterschiedliche Left-Shifts.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Tabelle 3.3: DES Permutation Tabellen: Initial Permutation (IP) links, und inverse Initial Permutation IP^{-1} (rechts).

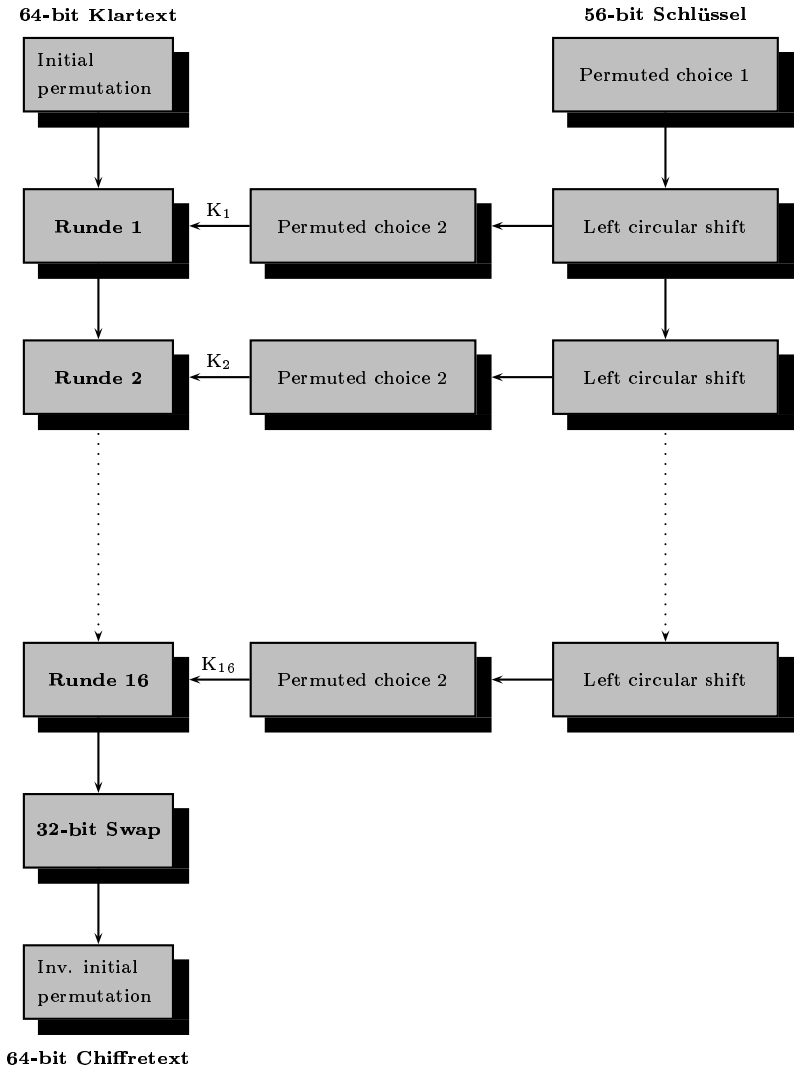


Abbildung 3.11: Der DES Verschlüsselungsalgorithmus

Die Startpermutation des DES Algorithmus und die invere Transformation sind durch fest vorgegebene Tabellen definiert. Diese sind in der Tabelle [3.3] aufgelistet. Um zu sehen, dass diese beiden Permutationen invers zueinander sind, betrachten wir folgenden 64 Bit Klartextstring:

$$\mathrm{IP} \ : \ \{\mathrm{M}_1 \ldots \mathrm{M}_{64}\} \longrightarrow \{\mathrm{M}_{\pi(1)} \ldots \mathrm{M}_{\pi(64)}\}$$

In der Tabelle [3.4] ist die Permutation der 64 Positionen des Bit Strings explizit dargestellt.

Wendet man die inverse Permutation auf diese Transformation an – diese ist im rechten Teil der Tabelle [3.3] aufgelistet, wird die ursprüngliche Reihenfolge der

M_1	$\longrightarrow$	M_{58}	M_{33}	$\longrightarrow$	M_{57}
M_2	$\longrightarrow$	M_{50}	M_{34}	$\longrightarrow$	M_{49}
M_3	$\longrightarrow$	M_{42}	M_{35}	$\longrightarrow$	M_{41}
M_4	$\longrightarrow$	M_{34}	M_{36}	$\longrightarrow$	M_{33}
M_5	$\longrightarrow$	M_{26}	M_{37}	$\longrightarrow$	M_{25}
M_6	$\longrightarrow$	M_{18}	M_{38}	$\longrightarrow$	M_{17}
M_7	$\longrightarrow$	M_{10}	M_{39}	$\longrightarrow$	M_9
M_8	$\longrightarrow$	M_2	M_{40}	$\longrightarrow$	M_1
M_9	$\longrightarrow$	M_{60}	M_{41}	$\longrightarrow$	M_{59}
M_{10}	$\longrightarrow$	M_{52}	M_{42}	$\longrightarrow$	M_{51}
M_{11}	$\longrightarrow$	M_{44}	M_{43}	$\longrightarrow$	M_{43}
M_{12}	$\longrightarrow$	M_{36}	M_{44}	$\longrightarrow$	M_{35}
M_{13}	$\longrightarrow$	M_{28}	M_{45}	$\longrightarrow$	M_{27}
M_{14}	$\longrightarrow$	M_{20}	M_{46}	$\longrightarrow$	M_{19}
M_{15}	$\longrightarrow$	M_{12}	M_{47}	$\longrightarrow$	M_{11}
M_{16}	$\longrightarrow$	M_4	M_{48}	$\longrightarrow$	M_3
M_{17}	$\longrightarrow$	M_{62}	M_{49}	$\longrightarrow$	M_{61}
M_{18}	$\longrightarrow$	M_{54}	M_{50}	$\longrightarrow$	M_{53}
M_{19}	$\longrightarrow$	M_{46}	M_{51}	$\longrightarrow$	M_{45}
M_{20}	$\longrightarrow$	M_{38}	M_{52}	$\longrightarrow$	M_{37}
M_{21}	$\longrightarrow$	M_{30}	M_{53}	$\longrightarrow$	M_{29}
M_{22}	$\longrightarrow$	M_{22}	M_{54}	$\longrightarrow$	M_{21}
M_{23}	$\longrightarrow$	M_{14}	M_{55}	$\longrightarrow$	M_{13}
M_{24}	$\longrightarrow$	M_6	M_{56}	$\longrightarrow$	M_5
M_{25}	$\longrightarrow$	M_{64}	M_{57}	$\longrightarrow$	M_{63}
M_{26}	$\longrightarrow$	M_{56}	M_{58}	$\longrightarrow$	M_{55}
M_{27}	$\longrightarrow$	M_{48}	M_{59}	$\longrightarrow$	M_{47}
M_{28}	$\longrightarrow$	M_{40}	M_{60}	$\longrightarrow$	M_{39}
M_{29}	$\longrightarrow$	M_{32}	M_{61}	$\longrightarrow$	M_{31}
M_{30}	$\longrightarrow$	M_{24}	M_{62}	$\longrightarrow$	M_{23}
M_{31}	$\longrightarrow$	M_{16}	M_{63}	$\longrightarrow$	M_{15}
M_{32}	$\longrightarrow$	M_8	M_{64}	$\longrightarrow$	M_7

Tabelle 3.4: DES Permutation Initial Permutation IP acting on 64-bit string.

Bits M_i rekonstruiert.

Details einer einzelnen Runde

Die Abbildung [3.12] zeigt die Details einer einzelnen Runde der DES Verschlüsselung. Wir betrachten zunächst die linke Seite dieser Abbildung. Die linke und rechte Hälfte jedes 64-Bit Zwischenresultats – diese sind mit L_{i-1} bzw. R_{i-1} bezeichnet – werden als separate 32-Bit Blöcke behandelt. Wie in jeder klassischen FEISTEL Chiffre kann die Grobstruktur der Blockverarbeitung durch folgende Gleichungen beschrieben werden:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus F(R_{i-1}, K_i) \end{aligned}$$

Der Rundenschlüssel hat eine Länge von 48 Bit. Die Länge des R-Blocks ist 32 Bit. Dieser wird – um mit dem Rundenschlüssel `XOR`-verknüpft zu werden

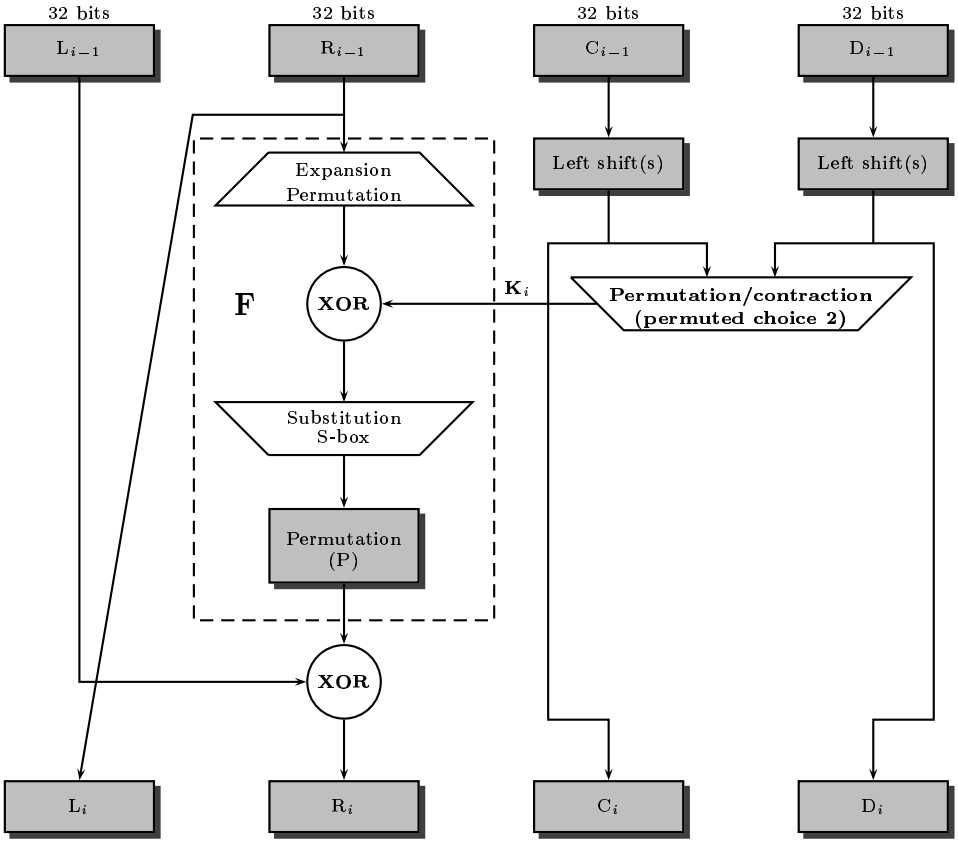


Abbildung 3.12: Die verschiedenen Schritte einer einzelnen Runde des DES Algorithmus.

– zunächst auf 48 Bit erweitert. Dies geschieht durch eine Expansions- bzw. Permutationsoperation in Form einer Tabelle. Die Expansion erweitert den 32 Bit Block um 16 Bit. Die **E/P** Operation ist in der Tabelle [3.5] aufgelistet.

Die resultierenden 48 Bits werden mit dem Rundenschlüssel K_i `XOR` verknüpft. Der Output der `XOR` Verknüpfung läuft in eine Substitutionsoperation. Diese Substitution wird durch insgesamt acht S-Boxen realisiert. Der Output der S-Boxen ist ein 32-Bit String, der anschließend durch eine weitere Permutation transformiert wird. Diese Permutation ist in der Tabelle [3.6] aufgeführt.

Die detaillierte Struktur der S-Boxen in der Rundenfunktion des DES Algorithmus ist in der Abbildung [3.13] dargestellt. Die Substitution wird durch acht S-Boxen realisiert, jede S-Box hat einen 6-Bit Input und produziert einen 4-Bit Output. Diese Transformationen sind in der Tabelle [3.7] explizit aufgelistet und sind folgendermaßen zu lesen:

1. Das erste und das letzte Bit des 6-Bit Inputs der S-Box S_i wird als binäre 2-Bit Zahl interpretiert.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Tabelle 3.5: Expansion/Permutations Operation

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Tabelle 3.6: Permutation Function P

2. Diese 2-Bit Zahl hat einen dezimalen Wertebereich von 0 bis 3 (dezimal).
3. Diese Zahl selektiert eine Zeile in der Box S_i für die Substitution.
4. Der mittlere 4-Bit Teil des 6-Bit Inputs selektiert eine spezielle Spalte der S-Box; diese Zahl hat einen Wertebereich zwischen 0 und 15.
5. Der Dezimalwert in dieser derart ausgewählten Zelle – festgelegt durch die Zeilen- und Spaltennummer – wird in den Binärwert umgewandelt, dieses Resultat ist der 4-Bit Output.

Beispiel:

Angenommen der Input der S-Box S_1 ist der Bitstring `011001`. Das erste und letzte Bit selektiert eine Zeile. In unserem Beispiel ist dies die Binärzahl `01`, das entspricht Zeile 1[13]. Die mittleren vier Bits sind `1100`, was der Dezimalzahl 12 entspricht. Daher wird Zeile 1 und Spalte 12 selektiert. In der S-Box S_1 in der Tabelle [3.7] findet man dort die Dezimalzahl 9 als Eintrag. Im Binärsystem entspricht dies dem String `1001`, dies ist dann der Output der Substitutionsoperation.

[13] Gezählt wird ab 0!.

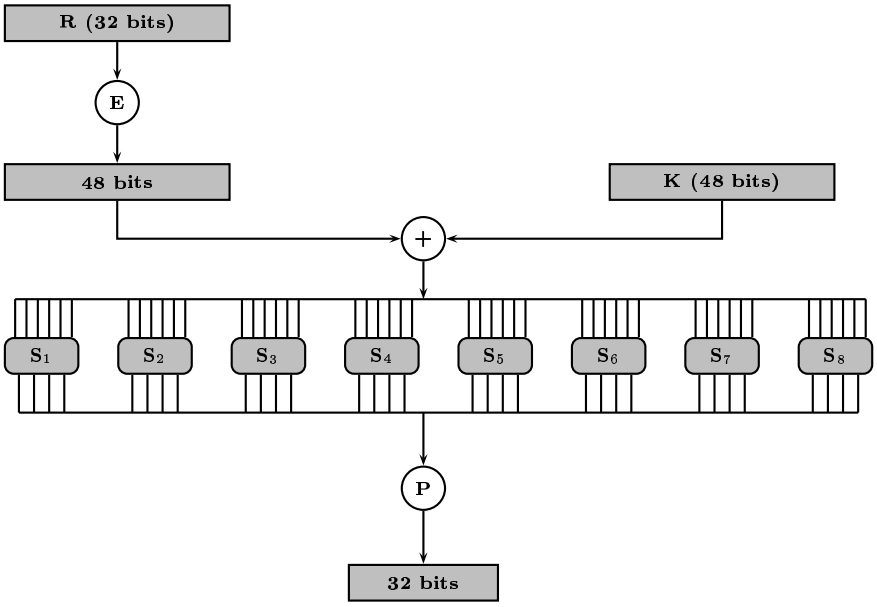


Abbildung 3.13: Berechnung von F(R,K).

$\mathbf{S}_1$	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
$\mathbf{S}_2$	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
$\mathbf{S}_3$	10	0	9	14	6	3	15	5	1	13	12	7	11	2	4	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
$\mathbf{S}_4$	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
$\mathbf{S}_5$	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
$\mathbf{S}_6$	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
$\mathbf{S}_7$	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
$\mathbf{S}_8$	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tabelle 3.7: Definition der DES S-Boxen.

DES Rundenschlüssel

Sieht man sich die Abbildungen [3.11] und [3.12] nochmals an, erkennt man, dass der 56 Bit Schlüssel zunächst durch eine Permutation läuft, die mit *permuted choice 1* bezeichnet wird. Die explizite Form dieser Operation ist in der Tabelle [3.8] aufgeführt.

Wie aus der Tabelle [3.8] zu sehen ist, treten die Zahlen 8, 16, 24, 32, 40, 48, 56 und 64 nicht auf. Mit anderen Worten, die Funktion *permuted choice 1* erwartet einen 64-Bit Schlüssel als Input, von diesen 64 Bit werden nur $56 = 6 \times 6$ Bits

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Tabelle 3.8: Permutations Tabelle der Operation *permuted choice 1*.

verwendet. Die restlichen acht Bits können als Parity Check Bits eingesetzt werden

Der Output der *permuted choice 1* Funktion wird in zwei 28 Bit Blöcke aufgeteilt, diese werden mit C_0 und D_0 bezeichnet. Um die verschiedenen Rundenschlüssel zu erhalten, wird auf jeden C_{j-1} und D_{j-1} separat eine zirkuläre Left-Shift Operation – diese heißt auch Rotation – ausgeführt. Diese Verschiebung ist rundenabhängig und variiert zwischen einem und zwei Bit.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabelle 3.9: Left shifts.

Die verschobenen Bit-Blöcke dienen einmal als Input in die folgende Runde, andererseits wird darauf die *permuted choice 2* Operation ausgeführt. Die explizite Form dieser Operation ist in der Tabelle [3.10] aufgelistet.

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Tabelle 3.10: Permutation Tabelle der Operation *permuted choice 2*.

Da der DES Algorithmus eine FEISTEL Chiffre darstellt, wird für die Dechiffrierung der gleiche Algorthmus verwendet wie für die Verschlüsselung mit der Ausnahme, dass die Reihenfolge der Rundenschlüssel umgekehrt wird.

Der Avalanche Effekt

Eine hochgradig erwünschte Eigenschaft jedes Verschlüsselungsalgorithmus ist, dass eine kleine Änderung sowohl im Klartext als im Schlüssel eine signifikante Änderung des Chiffretexts zur Folge hat. Insbesondere sollte eine Änderung eines Bits des Klartextblocks oder eine 1-Bit Änderung des Schlüssels die Änderung

vieler Bits des Chiffretexts zur Folge haben. Dies sind die Effekte der Diffusion und Konfusion. Sind die Anzahl dieser Änderungen klein, ist es für die Kryptoanalyse wesentlich einfacher, den möglichen Klartext bzw. Schlüssel einzugrenzen.

Der DES Algorithmus zeigt einen starken Avalanche Effekt. Die Tabelle [3.11] zeigt ein explizites Beispiel, das diesen Effekt demonstriert. In diesem Beispiel werden zwei Klartextblöcke mit DES verschlüsselt, die sich in einem Bit unterscheiden. Dies sind die Blöcke

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

mit dem Schlüssel:

0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

Die Tabelle [3.11] zeigt, dass bereits nach drei Runden die Verarbeitungsblöcke um 21 Bit unterscheiden. Nach der kompletten Verarbeitung (16 Runden) differieren die Blöcke in 34 Positionen. Die rechte Seite der Tabelle [3.11] zeigt einen ähnlichen Test mit dem folgenden 64-Bit Block als Input:

01101000 10000101 0010111 01111010 00010011 01110110 11101011 10100100

und zwei Schlüssel, die sich in nur einer Position unterscheiden:

1110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

0110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

Das Resultat zeigt erneut, dass Unterschiede in der Hälfte der Bits des Chiffreblocks auftreten und dass der Avalanche Effekt bereits nach wenigen Runden eintritt.

Change in plaintext		Change in Key	
Round	**# differing bits**	**Round**	**# differing bits**
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

Tabelle 3.11: Der Avalanche Effekt im Data Encryption Standard.

3.4 Betriebsmodi von Blockchiffren

Der DES Algorithmus war lange der grundlegende Baustein, um Datensicherheit zu realisieren. Um den DES Algorithmus in Anwendungen zu implementieren, sind vier mögliche Arbeitsmodi des Algorithmus definiert. Das Ziel dabei ist, buchstäblich alle mögliche Anwendungen abzudecken, für die DES als Verschlüsselungsalgorithmus angewendet werden kann. Wichtig zu realisieren ist, dass diese Arbeitsmodi der Blockchiffren nicht auf den DES Algorithmus beschränkt sind, sondern sie werden für alle symmetrischen Blockchiffren eingesetzt. Die vier Modi, die wir im Detail betrachten sind:

- ☞ Electronic Code Book Mode – ECB
- ☞ Cipher Block Chaining Mode – CBC
- ☞ Cipher Feedback Mode – CFB
- ☞ Output Feedback Mode – OFB

Die vier Arbeitsmodi sind vom NIST im Dokument FIPS PUB 81 im Dezember 1980 offiziell standardisiert [58]. Diese Modi können — wie erwähnt — auf *alle* symmetrischen Blockchiffren angewendet werden.

Neben diesen vier offizielln Betriebsmodi gibt es eine Reihe weiterer Arbeitsmodi von Blockchiffren.

Modus	Beschreibung	Typische Anwendung
Electronic Codebook ECB	Jeder 64 Bit Klartextblock wird unabhängig voneinander mit dem gleichen Key chiffriert.	Sichere Übertragung einzelner Werte e.g. ein Schlüssel
Cipher Block Chaining CBC	Der Input des Verschlüsselungs-algorithmus ist die XOR Ver - knüpfung der folgenden 64 Klartextbits und dem letzten 64 Bit Chiffreblock	Allgemeine blockorientierte Übertragung Authentifikation
Cipher Feedback CFB	J Bits der Eingabe werden in einem Vorgang verschlüsselt. Vorher erzeugter Chiffretext wird als Input des Verschlüsselungs-algorithmus benutzt, um pseudo-zufällige Ausgabe zu erzeugen. Diese wird mit Klartext geXORt, um die nächste Chiffretexteinheit zu erzeugen	Allgemeine blockorientierte Übertragung Authentifikation
Output Feedback OFB	Ähnlich wie CFB mit der Ausnahme, daß der Input des Verschlüsselungs-algorithmus der vorherige DES Output ist	Stromorientierte Übertragung über einen verrauschten Kanal wie z.B. Satelliten Kommunikation

Tabelle 3.12: Charakteristika verschiedener Arbeitsmodi von Blockchiffren

3.4.1 ECB–Mode

Die einfachste Arbeitsweise von DES[14] — das gilt für sämtliche symmetrischen Blockchiffren — ist der **electronic codebook mode** — kurz ECB Modus. In diesem Modus wird der Klartext in 64 Bit Blöcke aufgeteilt und jeder Block wird unabhängig von den anderen Blöcken verschlüsselt. Dabei wird immer der gleiche 56-Bit Key verwendet (bei DES). Dieses Verfahren ist in der Abbildung [3.14] dargestellt. Die Terminologie *codebook* wird hier verwendet, da — für einen festen Schlüssel — jedem 64 Bit Klartextblock eindeutig ein 64 Bit Chiffreblock zugewiesen wird. Daher kann man sich diese Verschlüsselungsart als gigantisches Codebuch vorstellen, das jedem der 2^{64} möglichen Klartextblock-Einträge (für festen Schlüssel) einen entsprechenden Chiffretextblock zuordnet.

Für Nachrichten, die länger als 64 Bit sind, wird der Klartextstring in 64 Bit Blöcke aufgesplittet und der letzte Block — falls notwendig — gepaddet. Bei der Entschlüsselung wird pro Zeiteinheit genau ein Block dechiffriert (siehe Abbildung [3.15]), wobei stets der gleiche Schlüssel verwendet wird. Dies bietet den Vorteil, dass die Entschlüsselung der Chiffretextblöcke in beliebiger Reihenfolge durchgeführt werden kann. In den Abbildungen [3.14] und [3.15] besteht der Klartext aus den 64 Bit Blöcken $P_1, P_2, \ldots, P_N$, die entsprechende Folge der Chiffreblöcke ist mit $C_1, C_2, \ldots, C_N$ bezeichnet.

[14] Siehe auch [74], Kapitel 7.5.3 oder die Diskussion in [81], Chap. 5.

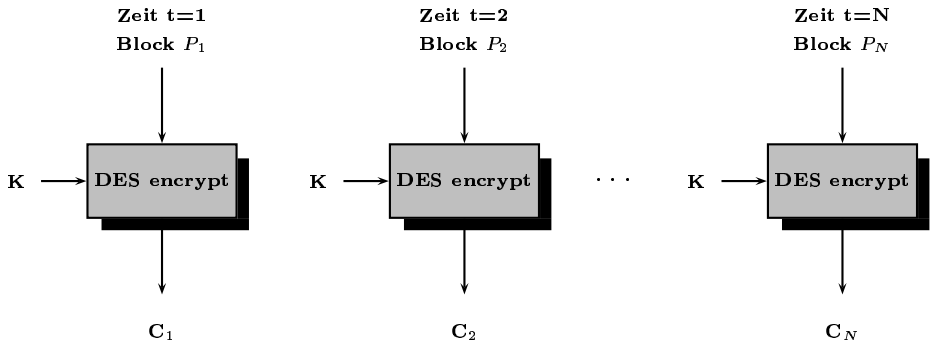


Abbildung 3.14: Electronic Codebook Mode – Verschlüsselung.

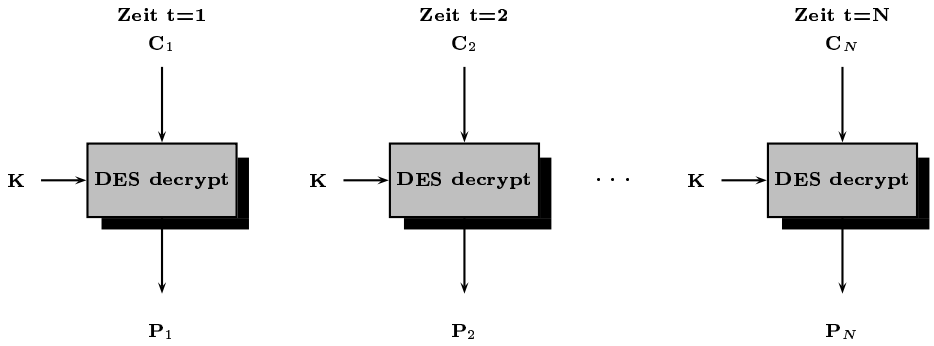


Abbildung 3.15: Electronic Codebook Mode – Entschlüsselung.

Der ECB Modus ist ideal geeignet für die Verschlüsselung kurzer Klartexte, beispielsweise eines Schlüssels. Falls daher ein DES Schlüssel sicher übertragen werden soll, ist der ECB Modus das geeignete Verfahren.

Die signifikante Eigenschaft des ECB Modes ist, dass der gleiche 64–Bit Block — falls dieser mehrfach in einem Klartext erscheint — immer auf den gleichen 64 Bit Chiffreblock abgebildet wird.

Für längere Nachrichten kann der Fall eintreten, dass der ECB Modus nicht mehr die gewünschte Sicherheit bietet. Im Fall, dass die Nachricht hochgradig strukturiert ist, kann ein Angreifer durch diese reguläre Struktur Rückschlüsse auf den Klartext ziehen. Ist beispielsweise bekannt, dass die Nachricht immer mit bestimmten vordefinierten Feldern beginnt, hat der Kryptoanalyst eventuell eine Reihe bekannter Klartext–Chiffretext–Paare, das die Analyse beträchtlich erleichtert. Falls die Nachricht Elemente hat, die sich in regelmäßigen Abständen wiederholen — beispielsweise mit einer Periode die ein Vielfaches von 64 Bit beträgt — können diese Elemente sehr leicht identifiziert werden.

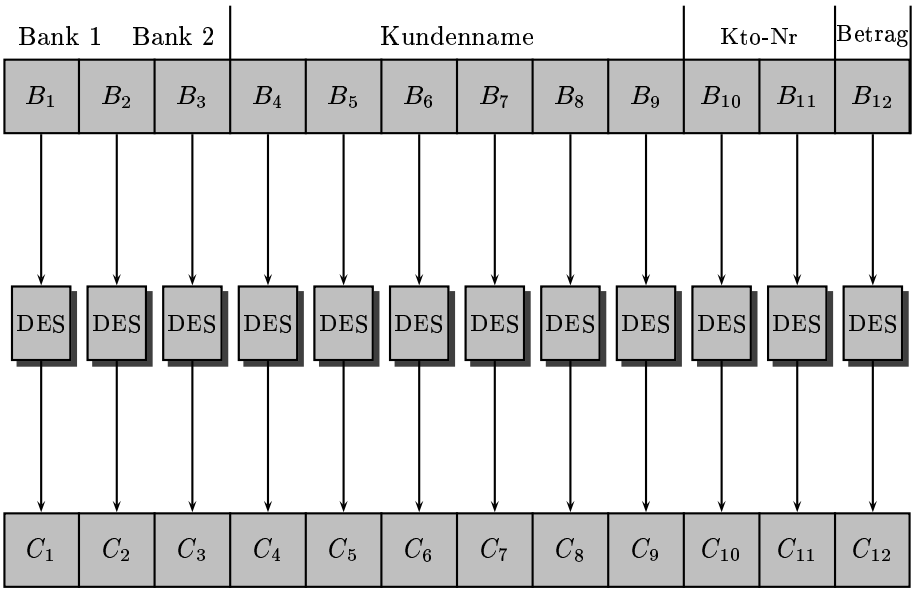
Beispiel [3.11]

Das folgende Beispiel aus [193, pp. 191–192] zeigt die Schwächen des ECB Modes sehr deutlich (siehe auch [74, p. 309]). Diese Szenario nennt man auch **Block-Wiederholung** bzw. **Block replay**.

Wir betrachten ein elektronisches Transfer-System zwischen zwei Banken, welches das Geld von einem Konto der einen Bank auf ein anderes Konto einer zweiten Bank transferiert. Um den Transfer zu vereinfachen, haben sich die beiden Banken auf ein Standardformt für den Nachrichtenaustausch geeinigt. Der Transfer wird elektronisch abgewickelt. Das Format des übertragenen Frames sieht folgendermaßen aus:

```
Bank 1:   Sender      1.5 Blöcke
Bank 2:   Empfänger   1.5 Blöcke
Kundenname            6 Blöcke
Konto-Nr.             2 Blöcke
Betrag                1 Block
```

Ein Block hat dabei die Länge von 8 Bytes. Die Nachrichten werden mittels irgendeiner symmetrischer Blockchiffre im ECB Modus chiffriert.



Mallory ist nun ein Angreifer, der die Kommunikation zwischen den beiden Banken abhört, Bank 1 von Alice und Bank 2 von Bob. Mit diesen zur Verfügung stehenden Informationen kann Mallory reich werden. Er muss dabei folgendermaßen vorgehen.

① Mallory richtet seinen Rechner so ein, daß alle verschlüsselten Nachrichten, die von Bank 1 an Bank 2 gesendet werden, aufgezeichnet werden.

② Dann transferiert Mallory $ 100 von einem Konto, das er sich auf Alices Bank eingerichtet hat auf ein anderes Konto, das er sich auf Bobs Bank

eingerichtet hat.

③ Kurze Zeit später führt Mallory diese Überweisung mit dem gleichen Betrag von 100 $ nochmals durch.

④ Anschließend untersucht Mallory die abgefangenen Nachrichten. Da der Klartext der beiden Überweisungen identisch ist — sie enthalten also gleiche Einträge in den Feldern Bank 1, Bank 2, Kundenname, Kto–Nr. und Betrag — und das Frame im ECB Modus verschlüsselt wurde, wurden dadurch zwei identische, chiffrierte Frames von Bank 1 an Bank 2 übertragen. Wenn nun der Angreifer Mallory die abgefangenen Nachrichten durchsucht, müssen unter denjenigen Nachrichten, die doppelt auftreten, seine beiden Überweisungen sein.

⑤ Wenn nun mehrere doppelte Nachrichten abgefangen wurden — das tritt in der Realität durchaus öfters auf — weiß Mallory immer noch nicht genau, welches Chiffrat seine Nachricht darstellt. Daher führt er nochmals eine Überweisung von 100 Dollar von seinem Bank 1 Konto auf sein Bank 2 Konto aus.

⑥ Vorausgesetzt, der Key wurde nicht gewechselt, muss Mallory nun drei identische, chiffrierte Nachrichten auf seinem Rechner haben. Diese drei Nachrichten sind genau seine eigene Überweisung.

⑦ Hat nun Mallory seine Message auf diese Weise identifiziert, kann er diese Nachricht nach eigenem Gutdünken in die Kommunikationsleitung einschleusen. Jedesmal, wenn er die Nachricht an Bank 2 sendet, werden auf seinem Konto 100 $ gutgeschrieben.

Dieser Angriff wird erst dann bemerkt, wenn die beiden Banken ihre Kontenbewegungen abgleichen. Dies wird in der Regel abends nach Geschäftsschluß geschehen. Dann werden diese Phantombewegungen entdeckt.[15]

Selbst wenn nun die beteiligten Banken das Format des Frames ändern, kann das System kompromittiert werden. Dadurch, dass das Frame beispielsweise einen Zeitstempel erhält, vermeidet man, dass Mallory das Paket identifiziert, das seiner Überweisung entspricht. In das zusätzliche Feld für den Zeitstempel trägt die absendende Bank jedesmal die Uhrzeit ein, daher können keine zwei identische Pakete entstehen. Das modifizierte Frameformat hat beispielsweise folgende Form:

```
Zeitstempel          1 Block
Bank 1:  Sender      1.5 Blöcke
Bank 2:  Empfänger   1.5 Blöcke
Kundenname           6 Blöcke
Konto-Nr.            2 Blöcke
Betrag               1 Block
```

Selbst solch eine Maßnahme verhindert nicht, dass Mallory auf nicht legitime Weise reich wird. Dabei wendet er eine Block replay Attacke an:

[15] Es bleibt daher genug Zeit für Mallory, sein Konto auf Bank 2 zu leeren und sich auf die Kaiman Inseln oder nach Argentien abzusetzen.

① Mallory fängt irgendeine zufällige Nachricht ab, die verschlüsselt von Bank 1 an die Bank 2 gesendet wird.

② Mallory schneidet die Blöcke 5 bis 12 der empfangenen (chiffrierten) Nachricht aus.

③ An deren Stelle platziert Mallory die Blöcke 5 bis 12, die für seinen eigenen Namen und seine Konto-Nummer stehen. Mallory ist schlau genug, das Betragsfeld *nicht* zu ändern! Dies ist ratsam, denn dadurch gewinnt Mallory Zeit, da für die Banken die Buchhaltung stimmt, d.h. der Angriff wird beim Abgleich der Kontenbewegungen nicht erkannt.

Diese Angriffsart wird erst dann erkannt, wenn einer der Kontoinhaber sich bei der Bank beschwert, oder die Bank 2 auf die ungewöhnlichen Aktivitäten des Kontos von Mallory aufmerksam wird. Spätestens dann wird es für Mallory Zeit, einen Flug nach Argentinien[16] zu buchen.

Die Kommunikationspartner können dieses Problem durch einen häufigeren Wechsel des Schlüssels minimieren. Dann muß Mallory nur schneller arbeiten, dies löst das Sicherheitsproblem daher nicht. Es gibt zwei Möglichkeiten, diese Sicherheitslücke zu schließen:

☞ Verwenden von Message Authenification Codes (MAC)

☞ Die Blockchiffre wird im **Chipher Block Chaining Modus** gefahren.

---

[16] Argentinien ist in diesem Fall zu empfehlen, da es kein Auslieferungsabkommen gibt.

3.4.2 CBC–Modus

Der **Chipher Block Chaining Modus** — oder kurz CBC-Mode genannt — ist so gestaltet, dass die Schwächen des ECB Modes umgangen werden. Das Ziel dieses Modus ist, unterschiedliche Chiffreblöcke zu generieren, wenn sich der Klartextblock wiederholt. Eine einfache Methode besteht in der Anwendung des *cipher block chaining* Modus. In diesem Schema ist der Input in den Verschlüsselungsalgorithmus die `XOR` Verknüpfung des aktuell zu verschlüsselnden Klartextblocks mit dem vorhergehenden Chiffreblock. Weiterhin wird immer der gleiche Schlüssel verwendet. Da der Input des Verschlüsselungsalgorithmus keine feste Relation zu dem Klartextblock hat, wird ein sich wiederholendes Muster im Klartext nicht auf den Chiffretext übertragen.

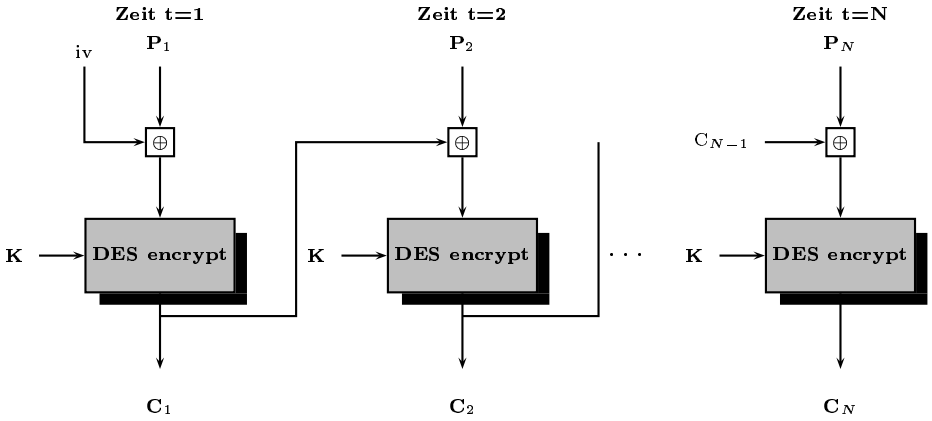


Abbildung 3.16: Verschlüsselung im CBC Modus.

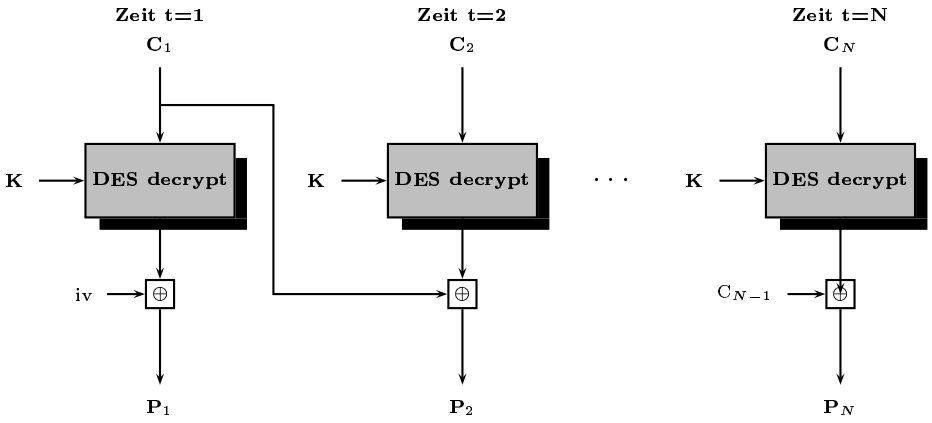


Abbildung 3.17: Cipher Block Chaining Mode – Entschlüsselung.

Bei der Entschlüsselung wird jeder Chiffreblock durch den Entschlüsselungsalgorithmus verarbeitet. Das Resultat wird mit dem vorhergehenden Chiffreblock

XOR-verknüpft um den Klartext zu erhalten. Um zu sehen, dass dies funktioniert, setzen wir:

$$C_n = E_k\ [\ C_{n-1} \oplus P_n\ ]$$

Damit:

$$\begin{aligned} D_k[C_n] &= D_k[E_k\ [\ C_{n-1} \oplus P_n\ ]] \\ &= C_{n-1} \oplus P_n \\ C_{n-1} \oplus D_k[C_n] &= C_{n-1}\ \oplus\ C_{n-1} \oplus P_n\ ]] \\ &= P_n \end{aligned}$$

Um den ersten Chiffretextblock zu erhalten, wird ein Initialisierungsvektor (IV) mit dem ersten Klaretxtblock geXORt. Bei der Entschlüsselung wird der Initialisierungsvektor benötigt, um im letzen Schritt (per XOR Verknüpfung) den ersten Klartextblock zu rekonstruieren.

Der Initialisierungsvektor muß dem Sender und dem Empfänger bekannt sein. Um diesen Grad an Sicherheit zu gewährleisten, muß der IV genauso geheim gehalten werden wie der Schlüssel. Dies kann beispielsweise dadurch geschehen, dass der IV im ECB Modus übertragen wird.

Zusammengefaßt kann man sagen, dass durch den Verkettungsmechanismus der CBC Modus das geeignete Verfahren ist, Nachrichten zu verschlüsseln, die länger als 64 Bit sind.

3.4.3 CFB–Mode

Das DES Verfahren ist im wesentlichen eine Block Chiffre, welches 64 Bit Blöcke transformiert. Es ist jedoch auch möglich, den Data Encryption Standard als Strom Chiffre einzusetzen, wenn man entweder im **Cipher Feedback Mode** (CFB) oder **Output Feedback Mode** (OFB) arbeitet. Bei der Verwendung einer Strom Chiffre ist es nicht mehr nötig, den Klartextstring auf eine Vielfaches einer bestimmten Blocklänge zu padden. Weiterhin operiert eine Strom Chiffre in Echtzeit. Wird also ein Zeichenstrom übertragen, wird jedes Zeichen mit einer zeichenorientierten Strom Chiffre verschlüsselt und sofort übertragen.

Eine gewünschte Eigenschaft einer Strom Chiffre ist, dass der Chiffretext von der gleichen Länge ist wie der Klartext. Werden also 8-Bit Zeichen übertragen, sollten auch 8 Bits verschlüsselt werden. Werden mehr als 8 Bit verwendet, wird unnötige Übertragungskapazität verbraucht.

Die Abbildung [3.18] zeigt das Prinzip des CFB Arbeitsmodus. In dieser Abbildung ist angenommen, dass die Übertragungseinheit j Bit sind, ein typischer Wert ist zum Beispiel $j = 8$. Wie beim CBC Modus werden die Klartexteinheiten verkettet. Daher ist der Chiffretext irgendeiner Klartexteinheit eine Funktion des gesamten vorausgehenden Klartexts.

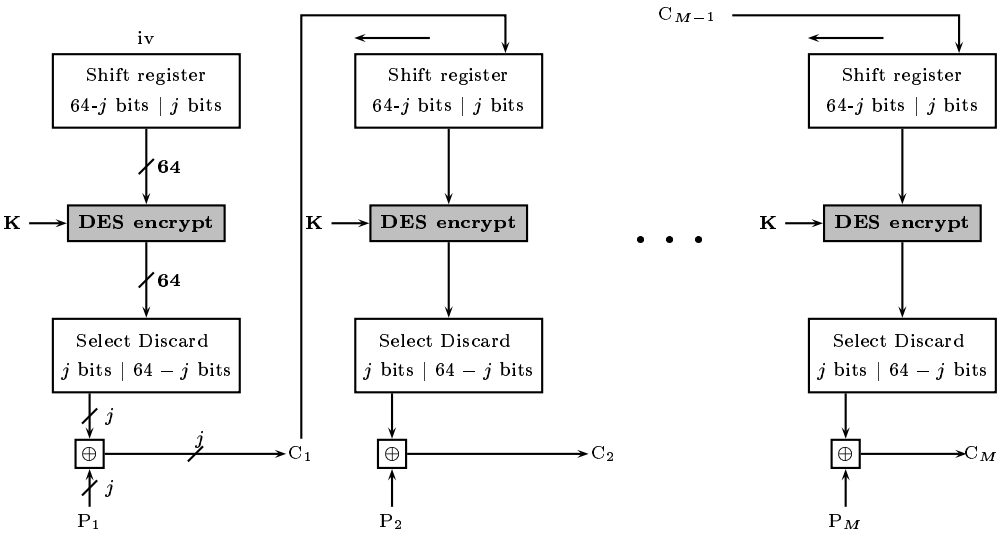


Abbildung 3.18: J-Bit Cipher Feedback Mode (CFB) – Verschlüsselung.

Wir betrachten zunächst die Verschlüsselung im CFB Modus (siehe Abbildung [3.18]). Der Input der Verschlüsselungsfunktion ist ein 64 Bit Shift Register, welches als Startwert einen Initialisierungsvektor IV enthält. Die j höchstwertigen Bit des Outputs des Verschlüsselungsalgorithmus werden anschließend mit der ersten Klartexteinheit P_1 `XOR` verknüpft. Dadurch entsteht die erste Chiffretexteinheit C_1. Dieser Block C_1 wird an den Empfänger der Nachricht übertragen. Gleichzeitig wird der Inhalt des Shift Registers um j Bit nach links verschoben

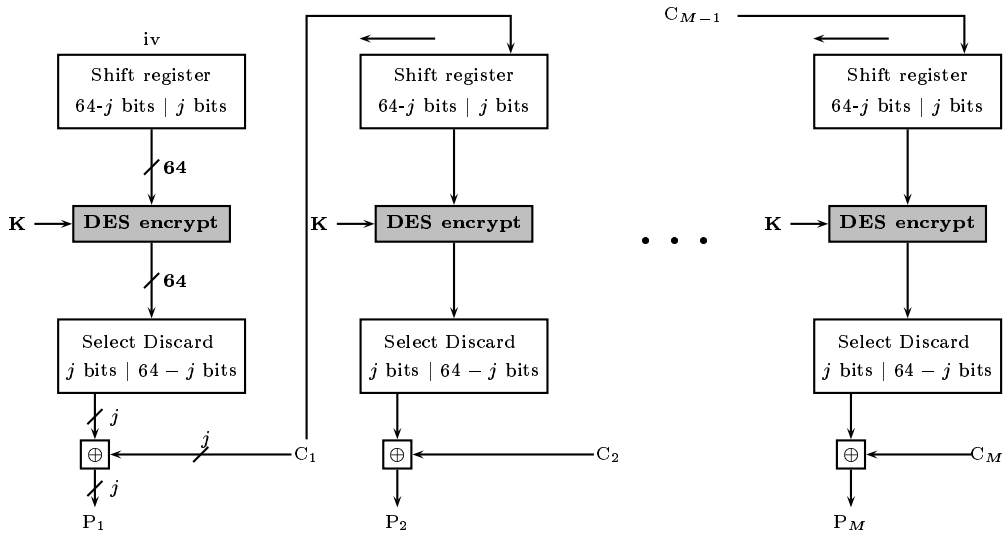


Abbildung 3.19: J-Bit Cipher Feedback Mode (CFB) – Entschlüsselung.

und der Chiffreblock aus j Bits in die j niedrigwertigen Stellen des Shiftregisters eingefügt. Dieses Procedere wird solange durchgeführt, bis sämtliche Klartexteinheiten chiffriert sind.

Für die Entschlüsselung wird das gleiche Verfahren angewendet wie für die Chiffrierung mit der Ausnahme, dass die empfangene Chiffretexteinheit mit dem Output des Verschlüsselungsalgorithmus `XOR` verknüpft wird um die Klartexteinheit zu reproduzieren. Wichtig an dieser Stelle ist zu sehen, dass die Verschlüsselungsfunktion – und *nicht* die Dechiffrierfunktion – verwendet wird. Der Grund dafür ist wie folgt zu sehen: Seien $\mathrm{S}_j(\mathrm{X})$ die signifikanten j Bits von X. Dann gilt gemäß dem Verschlüsselungsschema:

$$\mathrm{C}_1 = \mathrm{P}_1 \oplus \mathrm{S}_j(\mathrm{E}(IV))$$

Aufgrund der grundlegenden Eigenschaft der `xor` Operation muß gelten:

$$\mathrm{P}_1 = \mathrm{C}_1 \oplus \mathrm{S}_j(\mathrm{E}(IV))$$

Die gleiche Argumentation gilt auch für die nachfolgenden Schritte des Prozesses.

3.4.4 OFB–Modus

Der **Output Feedback Modus** (OFB) ist ähnlich strukturiert wie der CFB. Die Abbildung [3.20] zeigt das Schema des OFB Modus. Wie man dort erkennt wird der Output der DES Verschlüsselung in das Shift Register geschrieben, beim CFB Modus ist dies die Chiffretexteinheit, die in das Shiftregister geladen wird.

Ein Vorteil des OFB Modus ist darin zu sehen, dass Bit Fehler sich nicht fortpflanzen. Tritt beispielsweise während der Übertragung in einem verrauschten Kanal in der Chiffretexteinheit C_1 ein Fehler auf, ist davon nur der Klartextblock P_1 betroffen. Nachfolgende Klartexteinheiten sind davon nicht betroffen. Arbeitet man im CFB Modus, dann wird die Chiffretexteinheit C_1 in das Shiftregister geschrieben für die Verschlüsselung der nachfolgenden Einheit und verursacht deshalb zusätzliche Fehler.

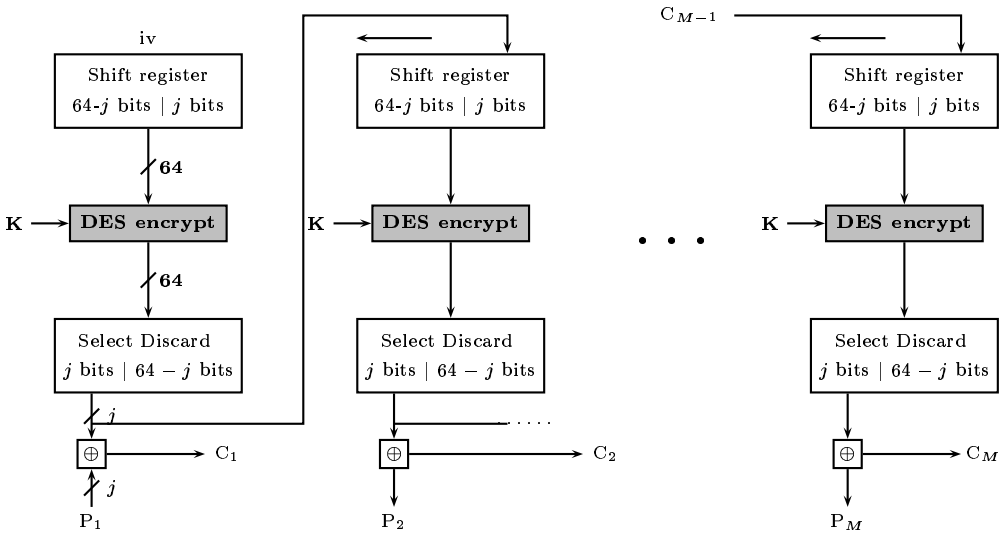


Abbildung 3.20: J-Bit Output Feedback Mode (OFB) – Verschlüsselung.

Der Nachteil des OFB Modus gegenüber dem CFB Modus liegt darin, dass diese Verschlüsselungsart angreifbar ist über eine sogenannte Message Stream Modifikation Attacke.

3.4.5 Der Counter Modus

Seit 2001 ist ein weiterer Arbeitsmodus für Blockchiffren vom NIST offiziell standardisiert ([72, 145]), der **Counter Mode**, kurz CTR Mode[17]. Der Counter Mode wandelt eine Block Chiffre in eine Strom Chiffre um. Das CTR-Verfahren generiert den folgenden Schlüsselstromblock durch die Chiffrierung aufeinanderfolgender Werte eines Zählers. Der Zähler selbst kann irgendeine einfache Funktion sein, die eine Folge generiert, die sich erst nach sehr vielen Werten wiederholt. Die einfachste und meist verbreitet Form dieser Funktion ist ein simpler Zähler.

[17] Siehe auch *R. Housley, Using Advanced Encryption Standard (AES) Counter Mode with IPSec Encapsulating Security Payload (ESP)*, RFC 3686.

3.5 Unix Passwörter

In diesem Abschnitt sehen wir uns eine Anwendung an, die den Data Encryption Standard in modifizierter Form nutzt. Das dabei angewendete Verfahren wird in UNIX Betriebssystemen genutzt, um das sicherheitsproblematische Speichern von Passwörten im Klartext zu umgehen. Das Verfahren ist in der Abbildung [3.21] skizziert.

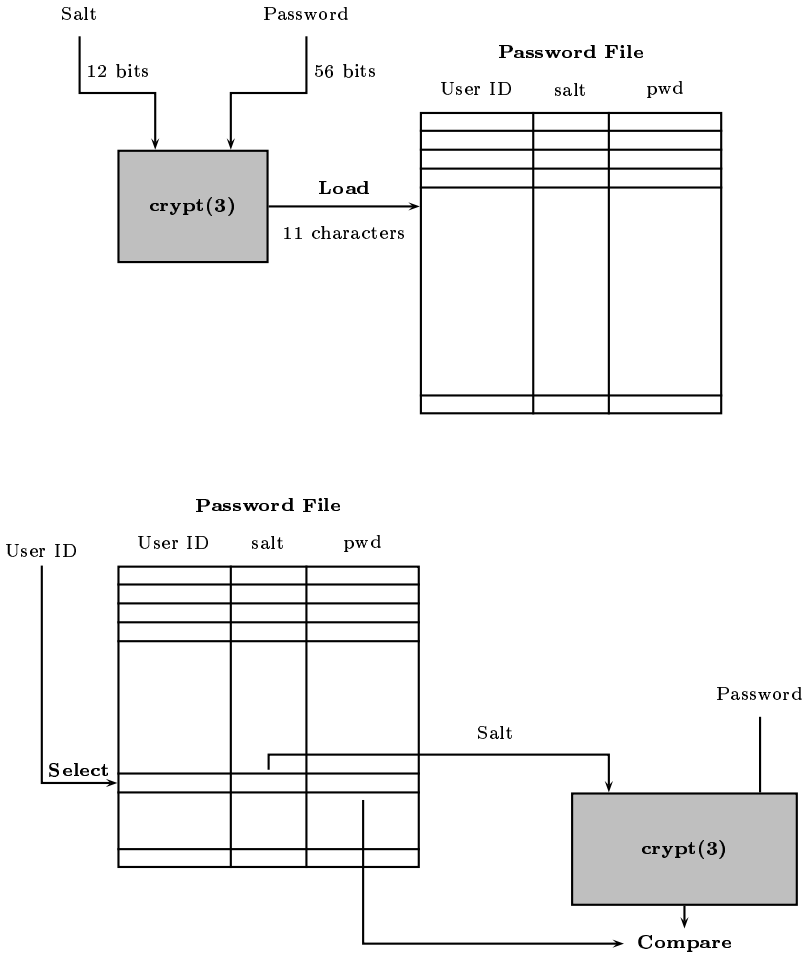


Abbildung 3.21: UNIX Password Scheme.

Jeder Benutzer eine UNIX Systems wählt ein Passwort, das aus acht druckbaren Zeichen besteht. Ist das Passwort länger, werden nur die erste acht Charakter verwendet. Diese Zeichen werden in eine 56 Bit String umgewandelt (dabei findet der 7 Bit ASCII Code Verwendung). Dieser 56 Bit String dient als Schlüssel einer Verschlüsselungsroutine, die unter `crypt(3)` bekannt ist. `crypt(3)` ba-

siert auf dem DES Algorithmus. Der DES Algorithmus ist aus bestimmten Gründen modifiziert, es wird ein sogenannter 12 Bit Salt-Wert benutzt. Typischerweise hängt dieser Wert vom Zeitpunkt ab, wann der Benutzer am System angemeldet wird – bzw. wann dem User das Passwort zugewiesen wird.

Der modifizierte DES Algorithmus operiert nun (mit dem Passwort als Schlüssel) auf einen 64 Bit Datenblock, der üblicherweise aus 64 0en besteht. Der Output dieser Verschlüsselung dient nun als Input einer zweiten Verschlüsselung mit dem modifizierten DES. Dieser Prozess wird insgesamt 25 mal durchgeführt. Der resultierende 64-Bit Output wird in eine Folge von 11 Zeichen umgesetzt. Dieses verschlüsselte Passwort wird dann zusammen mit einer Kopie des Salt-Wertes (dieser im Klartext) in der `passwd` Datei zusammen mit der UserID gespeichert.

Der Salt-Wert erfüllt drei Aufgaben:

- ☞ Der Salt-Wert verhindert, dass Passwort Duplikate in der Passwortdatei sichtbar sind. Selbst wenn zwei Benutzer zufällig das gleiche Passwort wählen, werden diese Passworte zu unterschiedlichen Zeiten dem jeweiligen Benutzer zugewiesen. Aus diesem Grund unterscheiden sich die um den Salt-Wert erweiterten Passwörter.
- ☞ Der Salt-Wert erhöht effektiv die Länge der Passwörter ohne dass sich die Benutzer die zusäztuliche Zeichen merken müssen. Damit wächst die Anzahl der mögliche Passwörter um den Faktor $2^{12} = 4.096$. Dadurch wird es für einen Angreifer auch schwieriger, ein Passwort zu erraten.
- ☞ Die Verwendung des Salt-Wertes verhindert die Verwendung einer Hardwareimplementierung des DES Algorithmus zur Durchführung einer sogenannten Brute Force Attacke.

Wenn sich nun ein User auf einem UNIX System einloggt, geschieht dies in zwei Schritte:

- ✘ er/sie identifiziert sich mit Hilfe der Eingabe einer UserID/Login Name (*Identifikation*
- ✘ er/sie beweist – er/sei authentifiziert sich – dass er/sie der angebliche Benutzer auch ist durch Eingabe des Passworts (Authentifikation)

Das Betriebssystem benutzt die eingegebene UserID als Schlüssel[18], um die Passwort Datei nach einem passenden Eintrag zu durchsuchen. Wird die UserID gefunden, wird das verschlüsselte Passwort und der Klartext-Salt-Wert ausgelesen. Die Benutzer-Passworteingabe und der ausgelesene Salt-Wert durchlaufen die `crypt(3)` Routine – und damit den Verschlüsselungsalgorithmus. Stimmt der Output dieser Operation mit dem gespeicherten, verschlüsselten Passwort überein, dann ist verifiziert, dass der Benutzer auch der ist, der er/sie behauptet zu sein.

Das Verschlüsselungsverfahren ist so designed um Angriffe abzuschrecken, die darin bestehen, Passwörter zu erraten.

[18] Schlüssel ist hier im Sinne der Datenbanktechnologie gemeint.

Kapitel 4

Weitere symmetrische Kryptosysteme

Heutzutage gibt es eine Vielzahl symmetrischer Kryptosysteme, die in kommerziellen Produkten implementiert sind. Es ist nicht möglich, alle Verfahren hier zu betrachten, es ist jedoch lohnenswert, sich einige Verfahren im Detail anzusehen[1].

4.1 International Data Encryption Algorithm

Der **International Data Encryption Algorithm** (IDEA) ist eine symmetrische Block Chiffre, die von X. LAI und J. MASSEY Ende der 80er Jahre entwickelt wurde. IDEA war eine der symmetrischen Kryptosysteme, die als Nachfolger des DES Verfahrens gehandelt wurde. IDEA ist in der Software PGP (Pretty Good Privacy) integriert.

4.1.1 Designprinzipien

IDEA ist eine Block Chiffre, die mit Hilfe eines 128 Bit Schlüssels Daten in 64 Bit Blöcken chiffriert. Die wichtigsten Prinzipien beim Design dieser Chiffre sind:

- **Blocklänge:**
  Die Blocklänge sollte so groß gewählt werden, dass eine statistische Analyse verhindert wird. Andererseits scheint die Komplexität der Implementierung einer effektive Verschlüsselungsfunktion exponentiell mit der Blocklänge zu wachsen. Die Verwendung einer Blocklänge mit 64 Bit wird allgemein als ausreichend stark angesehen.

- **Schlüssellänge**
  Die Länge des Schlüssels muß groß genug gewählt werden, um eine erschöpfende Schlüsselsuche in Form einer Brute Force Attacke zwecklos zu

[1] siehe dazu das Buch von BRUCE SCHNEIER [193], in dem zahlreiche Kryptosysteme diskutiert werden.

machen. Mit der Wahl von Schlüsseln der Länge von 128 Bit scheint IDEA bis weit in die Zukunft sicher zu sein.

- **Konfusion**:
  Der Chiffretext sollte vom Klartext und vom Schlüssel in einer komplexen Weise abhängen. Das Ziel, das man mit Konfusion erreichen will ist, den Zusammenhang der Statistik des Chiffretexts mit derjenigen des Klartexts zu verschleiern. IDEA erreicht dieses Ziel durch die Verwendung drei verschiedener Operationen. Dies muß man im Vergleich zum DES Algorithmus sehen, bei dem die Konfusion durch Verwendung der `XOR` Opertaion sowie kleiner, nichtlinearer S-Boxen erzielt wird.

- **Diffusion**:
  Jedes Bit des Klartexts sollte jedes Bit des Chiffretextes beeinflussen und jedes Bit des Schlüssel sollte ebenfalls jedes Bit des Chiffrats beeinflussen. Das Verteilen eines Klartextbits über möglichst viele Chiffretextbits versteckt die Statistik des Klartexts. IDEA setzt dieses Konzept sehr effektiv um.

Wir betrachten die Implementierung im IDEA der letzten beiden Punkte.

Im IDEA Kryptosystem wird Konfusion durch eine Mischung der folgenden drei Operationen erzielt. Jede dieser Operationen hat als Input zwei 16 Bit Blöcke und produziert einen 16 Bit Block.

- Bitweises `XOR`, bezeichnet durch $\oplus$
- Addition von Integerzahlen modulo 2^{16} (i.e. modulo 65.536); diese Operation wird mit $\boxplus$ bezeichnet. Der Input und der Output sind 16 Bit vorzeichenlose Ganzzahlen (unsigned int).
- Multiplikation von Integers modulo $2^{16}+1$ (i.e. modulo 65.537). Input und Output werden als vorzeichenlose 16-Bit Ganzzahlen behandelt. Ein 16-Bit Block aus lauter `0`en stellt die Zahl 2^{16} dar. Diese Operation wird mit $\odot$ bezeichnet.

Diese drei Operation sind im folgenden Sinne inkompatibel:

1. Kein Paar dieser drei Operationen erfüllt das Distributivgesetz, e.g.
$$a \boxplus (b \odot c) \neq (a \boxplus b) \odot (a \boxplus c)$$
2. Kein Paar dieser drei Operationen erfüllt ein Assoziativgesetz:
$$a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c$$

Beispiele:

Um zu verstehen, wie diese Operationen funktionieren, sehen wir uns einige Verknüpfungen von 2-Bit Zahlen an (anstelle von 16-Bit Zahlen). Die Ergebnisse sind in der Tabelle [4.1] zusammengefaßt.

`XOR` operiert bitweise mit:

$$0 \oplus 0 = 0; \quad 0 \oplus 1 = 1 \oplus 0 \text{ and } 1 \oplus 1 = 0$$

Die Addition modulo $2^2 = 4$ ist beispielsweise:

$$10 \boxplus 11 = 2 + 3 = 5 \equiv 2 \bmod 2^2 = 10$$

A	**B**	**A** $\oplus$ **B**	**A** $\boxplus$ **B**	**A** $\odot$ **B**
00	00	00	00	01
00	01	01	01	00
00	10	10	10	11
00	11	11	11	10
01	00	01	01	00
01	01	00	10	01
01	10	11	11	10
01	11	10	00	11
10	00	10	10	11
10	01	11	11	10
10	10	00	00	00
10	11	01	01	01
11	00	11	11	10
11	01	10	00	11
11	10	01	01	01
11	11	00	10	00

Tabelle 4.1: Die drei grundlegenden Verknüpfungen, die im IDEA benutzt werden.

Die Multiplikation modulo 5 operiert folgendermaßen, ein String mit 0en stellt die Zahl $2^2 = 4$ dar:

$$
\begin{aligned}
00 \oplus 00 &= 4 \cdot 4 = 16 \equiv 1 \bmod 5 = 01 \\
00 \oplus 01 &= 4 \cdot 1 = 4 \equiv 4 \bmod 5 = 00 \\
00 \oplus 10 &= 4 \cdot 2 = 8 \equiv 3 \bmod 5 = 11 \\
00 \oplus 11 &= 4 \cdot 3 = 12 \equiv 2 \bmod 5 = 10 \\
01 \oplus 00 &= 1 \cdot 4 = 4 \equiv 4 \bmod 5 = 00 \\
01 \oplus 01 &= 1 \cdot 1 = 1 \equiv 4 \bmod 5 = 01 \\
01 \oplus 10 &= 1 \cdot 2 = 2 \equiv 3 \bmod 5 = 10 \\
01 \oplus 11 &= 1 \cdot 3 = 3 \equiv 2 \bmod 5 = 11 \\
10 \oplus 00 &= 2 \cdot 4 = 8 \equiv 3 \bmod 5 = 11 \\
10 \oplus 01 &= 2 \cdot 1 = 2 \equiv 2 \bmod 5 = 10 \\
10 \oplus 10 &= 2 \cdot 2 = 4 \equiv 4 \bmod 5 = 11 \\
10 \oplus 11 &= 2 \cdot 3 = 6 \equiv 1 \bmod 5 = 01 \\
11 \oplus 00 &= 3 \cdot 4 = 12 \equiv 2 \bmod 5 = 10 \\
11 \oplus 01 &= 3 \cdot 1 = 3 \equiv 3 \bmod 5 = 11 \\
11 \oplus 10 &= 3 \cdot 2 = 6 \equiv 1 \bmod 5 = 01 \\
11 \oplus 11 &= 3 \cdot 3 = 9 \equiv 4 \bmod 5 = 00
\end{aligned}
$$

Die Verwendung der Kombination dieser drei Operationen hat eine komplexe Transformation des Inputs zur Folge, was eine kryptoanalytische Untersuchung viel schwerer macht als die Anwendung der `XOR` Operation bei dem DES Algorithmus.

Das IDEA Verfahren realisiert die Diffusion durch die grundlegenden Bauelemente des Algorithmus, die unter dem Begriff **Multiplikation/Additions Struktur** (MA) bekannt sind.

Diese Struktur hat als Input zwei 16 Bit Blöcke (F_1, F_2), die sowohl vom Klartext als auch von zwei 16 Bit Teilschlüssel (Z_5, Z_6) abgeleitet werden. Die MA-Struktur erzeugt zwei 16 Bit Outputs (G_1, G_2). Wie aus der Abbildung [4.1] ersichtlich ist, gilt:

$$
\begin{aligned}
G_2 &= (F_2 \boxplus (F_1 \odot Z_5)) \odot Z_6 \\
G_1 &= (F_1 \odot Z_5) \boxplus G_2
\end{aligned}
$$

Ein erschöpfender Computer Test hat gezeigt, dass jedes Output-Bit der ersten Runde von jedem Bit der aus dem Klartext abgeleiteten Eingaben und jedem Bit der Teilschlüssel abhängt. Diese spezielle Struktur wird im IDEA Algorithmus acht mal durchgeführt, was auf eine hochgradige Diffusion hinausführt.

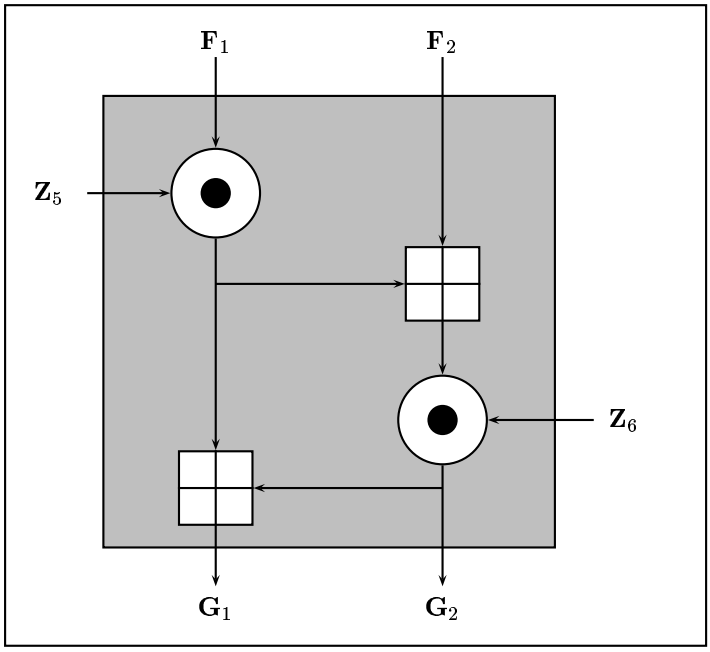


Abbildung 4.1: Multiplikation/Additions Struktur des IDEA Algorithmus.

4.1.2 IDEA Verschlüsselung

Das generelle Schema der IDEA Verschlüsselung ist in der Abbildung [4.2] dargestellt. Wie bei allen Verschlüsselungsverfahren hat man zwei Inputs, den zu verschlüsselnden Klartext und den Schlüssel. Der Klartextblock hat eine Länge von 64 Bit, der Schlüssel eine Länge von 128 Bit.

Die linke Seite der Abbildung [4.2] zeigt, dass der IDEA Verschlüsselungsalgorithmus aus acht Runden besteht, auf die eine abschließenden Runde mit einer Output Transformation folgt. Der Algorithmus spaltet den Input in vier 16-Bit Blöcke auf. Jede der acht (+1) Runden hat vier 16 Bit Blöcke als Input und produziert vier 16 Bit Blöcke Output. Nach der abschließenden Runde werden diese vier Teilblöcke zu einem 64 Bit Chiffreblock konkateniert. Zusätzlich gehen in jede der acht Runden sechs 16 Bit Teilschlüssel ein, die letzte Runde verwendet vier Subkeys. Insgesamt arbeitet der Algorithmus mit 52 16 Bit Teilschlüssel. Die rechte Seite der Abbildung [4.2] deutet an, dass diese 52 Teilschlüssel aus dem ursprünglichen 128 Bot Schlüssel abgeleitet werden.

Details der IDEA Verschlüsselung

Sehen wir uns den Algorithmus einer einzelnen Runde im Detail an; die erste Runde des IDEA Algorithmus ist in der Abbildung [4.3] dargestellt, Die Runde beginnt mit einer Transformation, die die vier Teilblöcke mit vier Teilschlüssel verknüpft. Dabei wird die Addition modulo 2^{16} und Multiplikation modulo $2^{16}+1$ verwendet. Die Schritte sind:

1. Multipliziere X_1 und den ersten Teilschlüssel Z_1 modulo $2^{16}+1$.

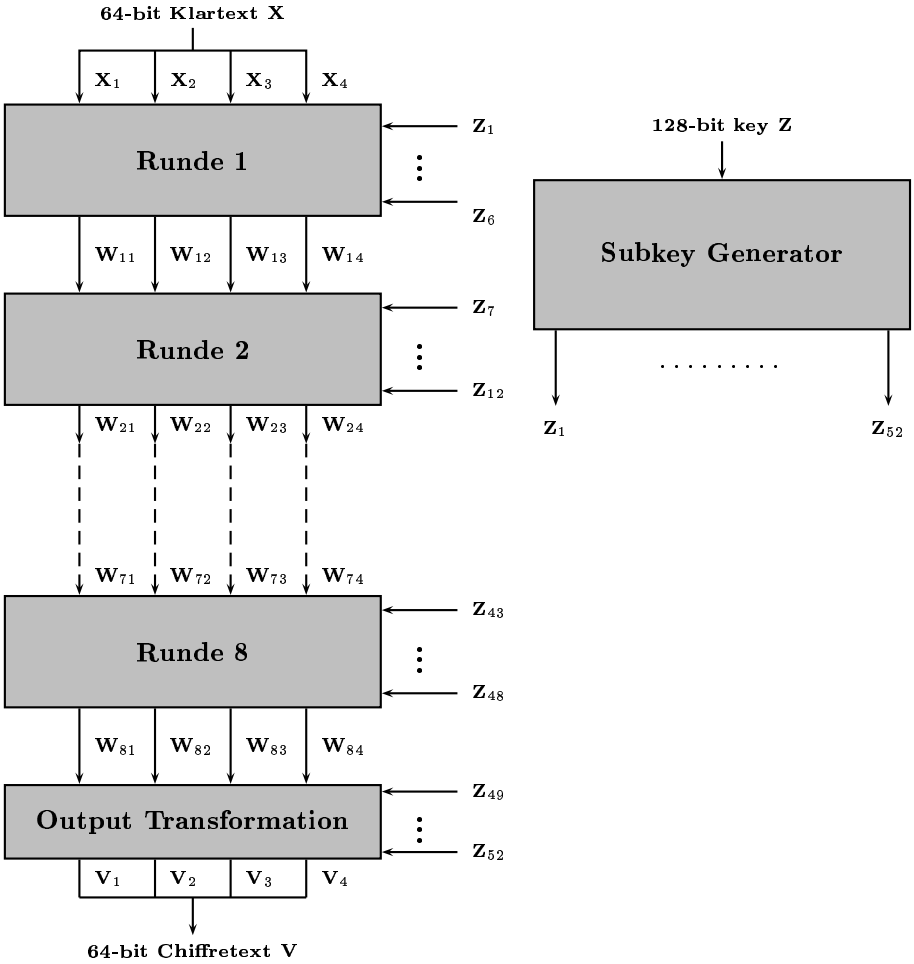


Abbildung 4.2: Struktur des IDEA Verschüsselungs Schemas.

2. Addiere X_2 und den zweiten Teilschlüssel Z_2 modulo 2^{16}.
3. Addiere X_3 und den dritten Teilschlüssel Z_3 modulo 2^{16}.
4. Multipliziere X_4 mit dem vierten Teilschlüssel Z_4 modulo $2^{16}+1$.
5. XOR die Resultate der Schritte (1) und (3).
6. XOR die Resultate der Schritte (2) und (4).
   Der Output der Schritte (5) und (6) bilden den Input der MA Struktur.
7. Multipliziere den Output von Schritt (5) mit dem fünften Teilschlüssel Z_5 modulo $2^{16}+1$.
8. Addiere die Ergebnisse der Schritte (6) und (7) modulo 2^{16}.
9. Multipliziere den Output von Schritt (8) mit dem sechsten Teilschlüssel Z_6 modulo $2^{16}+1$.

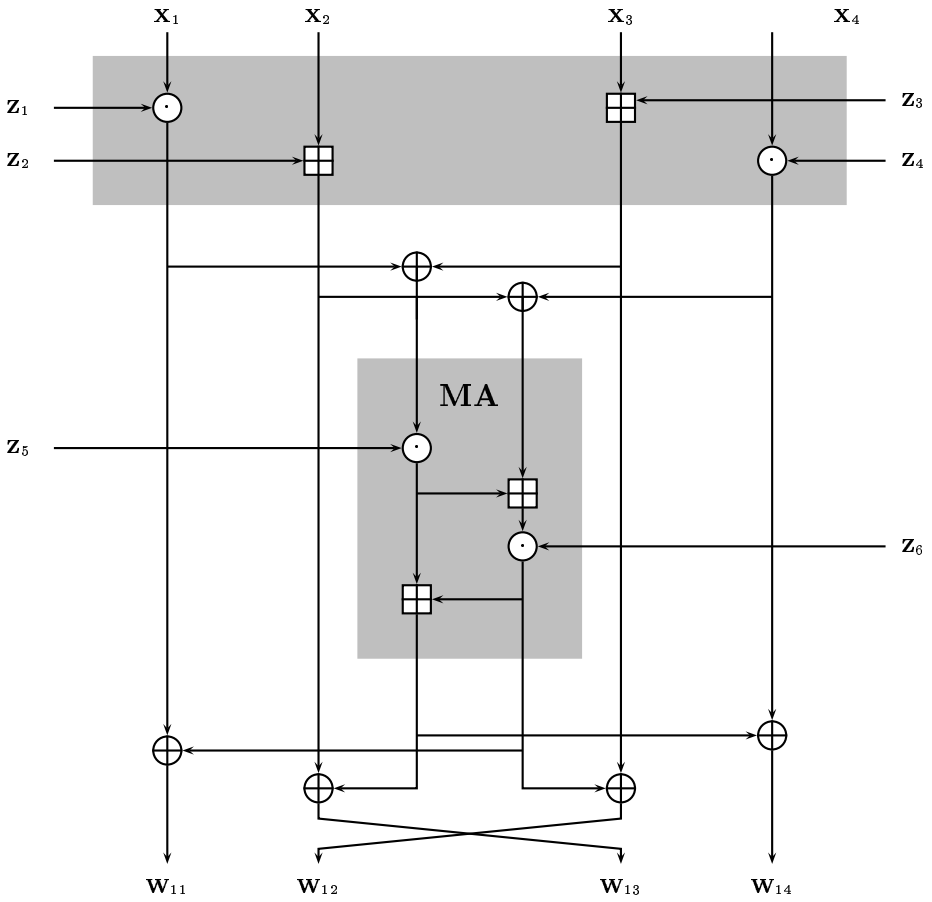


Abbildung 4.3: Skizze einer einzelnen IDEA Verschlüsselungs-Runde.

10. Addiere die Resultate der Schritte (7) und (9) modulo 2^{16}.
11. XOR die Resultate der Schritte (1) und (9).
12. XOR die Resultate der Schritte (3) und (9).
13. XOR die Resultate der Schritte (2) und (10).
14. XOR die Resultate der Schritte (4) und (10).
15. Bezeichne den Output von Schritt (11) mit W_{11}.
16. Bezeichne den Output von Schritt (12) mit W_{13}.
17. Bezeichne den Output von Schritt (13) mit W_{12}.
18. Bezeichne den Output von Schritt (14) mit W_{17}.

Der Output der Runde sind vier 16 Bit Teilblöcke, die die Resultate der Schritte (11), (12), (13) und (14) sind. Die beiden inneren Blöcke werden vertauscht; diese vier Blöcke bilden den Input der nächsten Runde.

Die letzte Runde des IDEA Algorithmus ist in der Abbildung [4.4] dargestellt. Diese hat eine ähnliche Struktur wie der obere grau hinterlegte Bereich der vorhergehenden Runden (siehe Abbildung [4.3]). Der Unterschied liegt darin, daß der zweite und dritte Input vertauscht wird bevor diese Blöcke in die Verarbeitungseinheit einlaufen. Dies hat den Effekt, die Vertauschung am Ende der achten Runde rückgängig zu machen. Der Grund für diese Operation liegt darin, daß die Entschlüsselung die gleich Struktur hat wie die Verschlüsselung. Die letzte Transformation benötigt lediglich vier Teilschlüssel. Die letzten Schritte sind:

1. Multipliziere W_{81} und den Teilschlüssel Z_{49} modulo $2^{16}+1$.
2. Addiere W_{83} und den Teilschlüssel Z_{50} modulo 2^{16}.
3. Addiere W_{82} und den Teilschlüssel Z_{51} modulo 2^{16}.
4. Multipliziere W_{84} und den Teilschlüssel Z_{52} modulo $2^{16}+1$.

Schließlich werden die resultierenden vier 16-Bit Blöcke konkateniert und liefern den Chiffreblock mit der Länge von 64 Bit.

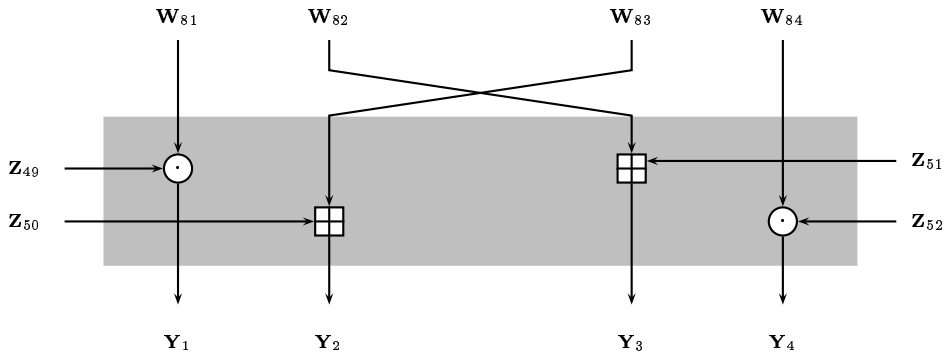


Abbildung 4.4: Output Transformation des IDEA Algorithmus.

Erzeugung der Teilschlüssel

Wie in der Abbildung [4.2] dargestellt ist, werden insgesamt 52 Subkeys der Länge von je 16 Bit aus dem 128 Bit Schlüssel erzeugt. Der Algorithmus zur Erzeugung dieser Teilschlüssel besteht aus folgenden Einzelschritten: algorithm works as follows:

- Der 128-Bit Schlüssel wird zunächst in acht 16-Bit Schlüssel aufgeteilt. Dieser Schritt generiert die Keys:

  $$Z_1, Z_2, \ldots, Z_8$$

  Die ersten sechs dieser Schlüssel werden in der ersten Runde benutzt, die verbleibenden beiden Teilschlüssel Z_7 und Z_8 in der zweiten Runde.

- Then the 128-bit key is left-shifted by 25 bits
- Again the 128-bit key is divided into eight 16-bit subkeys producing the keys $Z_9, Z_{10}, \ldots, Z_{16}$. The first four of these keys are used in round 2, the last four in round 3.
- Rotate the 128-bit key another 25 bits to the left.
- Divide the resulting 128-bit string into eight subkeys $Z_{17}, \ldots, Z_{24}$.
- Continue until all 52 subkeys are generated.

4.1.3 IDEA Dechiffrierung

Das Entschlüsselungsverfahren ist im wesentlichen das gleiche wie der Verschlüsselungsprozess. Die Entschlüsselung erhält man, indem das Chiffrat als Input der gleichen IDEA Struktur verwendet wird (siehe Abbildung [4.2]). Der Unterschied liegt in der Wahl der Subkeys. Diese werden mit $U_1, U_2, \ldots, U_{52}$ bezeichnet und können aus den Subkeys der Verschlüsselung folgendermaßen abgeleitet werden:

1. Die ersten vier Subkeys der Entschlüsselungsrunde i werden aus den ersten vier Subkeys der Verschlüsselungsrunden $(10 - i)$ abgeleitet, wobei der letzte Transformationsschritt als 9 zählt. the last transformation stage is counted as 9. The first and fourth decryption subkeys are equal to the multiplicative inverse modulo $(2^{16} + 1)$ of the corresponding first and fourth encryption subkeys. For rounds 2 through 8, the second and third decryption subkeys are equal to the additive inverse modulo 2^{16} of the corresponding third and second encryption subkeys. For rounds 1 and 9. the second and third decryption subkeys are equal to the additive inverse modulo 2^{16} of the corresponding second and third encryption subkeys.
2. For the first eight rounds, the last two subkeys of decryption round i are equal to the last two subkeys of encryption round $(9 - i)$.

Diese Beziehungen sind in der Tabelle [4.2] zusammengefaßt.

For the multiplicative inverse we use the notation Z_j^{-1} such that the relation

$$Z_j \odot Z_j^{-1} = 1$$

holds. Now, $2^{16}+1$ is a prime number[2], therefore, each nonzero integer $Z_j \leq 2^{16}$ is coprime to $2^{16} + 1$ and therefore has a unique multiplicative inverse modulo $2^{16} + 1$ For the additive inverse, we use the notation $-Z_j$ such that

$$-Z_j \boxplus Z_j = 0$$

Um zu verifizieren, dass der gleiche Algorithmus mit den Entschlüsselungs-Subkeys den korrekten Klartext produziert, betrachten wir die Abbildung [4.5],

[2] Recall that this is one of FERMAT's numbers.

Stage	E.-Subkeys	D.-Subkeys	Equivalence
R 1	Z_1, Z_2, Z_3 Z_4, Z_5, Z_6	U_1, U_2, U_3 U_4, U_5, U_6	$Z_{49}^{-1}, -Z_{50}, -Z_{51}$ $Z_{52}^{-1}, Z_{47}, Z_{48}$
R 2	Z_7, Z_8, Z_9 Z_{10}, Z_{11}, Z_{12}	U_7, U_8, U_9 U_{10}, U_{11}, U_{12}	$Z_{43}^{-1}, -Z_{45}, -Z_{44}$ $Z_{46}^{-1}, Z_{41}, Z_{42}$
R 3	Z_{13}, Z_{14}, Z_{15} Z_{16}, Z_{17}, Z_{18}	U_{13}, U_{14}, U_{15} U_{16}, U_{17}, U_{18}	$Z_{37}^{-1}, -Z_{39}, -Z_{38}$ $Z_{40}^{-1}, Z_{35}, Z_{36}$
R 4	Z_{19}, Z_{20}, Z_{21} Z_{22}, Z_{23}, Z_{24}	U_{19}, U_{20}, U_{21} U_{22}, U_{23}, U_{24}	$Z_{31}^{-1}, -Z_{33}, -Z_{32}$ $Z_{34}^{-1}, Z_{29}, Z_{30}$
R 5	Z_{25}, Z_{26}, Z_{27} Z_{28}, Z_{29}, Z_{30}	U_{25}, U_{26}, U_{27} U_{28}, U_{29}, U_{30}	$Z_{25}^{-1}, -Z_{27}, -Z_{26}$ $Z_{28}^{-1}, Z_{23}, Z_{24}$
R 6	Z_{31}, Z_{32}, Z_{33} Z_{34}, Z_{35}, Z_{36}	U_{31}, U_{32}, U_{33} U_{34}, U_{35}, U_{36}	$Z_{19}^{-1}, -Z_{21}, -Z_{20}$ $Z_{22}^{-1}, Z_{17}, Z_{18}$
R 7	Z_{37}, Z_{38}, Z_{39} Z_{40}, Z_{41}, Z_{42}	U_{37}, U_{38}, U_{39} U_{40}, U_{41}, U_{42}	$Z_{13}^{-1}, -Z_{15}, -Z_{14}$ $Z_{16}^{-1}, Z_{11}, Z_{12}$
R 8	Z_{43}, Z_{44}, Z_{45} Z_{46}, Z_{47}, Z_{48}	U_{43}, U_{44}, U_{45} U_{46}, U_{47}, U_{48}	$Z_7^{-1}, -Z_9, -Z_8$ Z_{10}^{-1}, Z_5, Z_6
	Z_{49}, Z_{50} Z_{51}, Z_{52}	U_{49}, U_{50} U_{51}, U_{52}	$Z_1^{-1}, -Z_2$ $-Z_3, Z_4^{-1}$

Tabelle 4.2: Schlüssel zur Ver- bzw. Entschlüsselung im IDEA Algorithmus

die den Verschlüsselungsprozess auf der linken Seite (abwärts) und den Dechiffierungsprozess auf der rechten Seite von unten nach oben zeigt. Jede der acht Runden ist in zwei Teilschritte aufgeteilt, die man *sub-encryption* und *transformation* nennt. Der Transformation-Schritt entspricht dem oberen schattierten Teil in der Abbildung [4.3], der Subencryption Schritt bezieht sich auf den Rest der verarbeitungsrunde.

Betrachte die unteren Boxen der Verschlüsselungs- und der Entschlüsselungskette. Auf der verschlüsselunsgseite haben wir:

$$
\begin{aligned}
Y_1 &= W_{81} \odot Z_{49} \\
Y_2 &= W_{83} \boxplus Z_{50} \\
Y_3 &= W_{82} \boxplus Z_{51} \\
Y_4 &= W_{84} \odot Z_{52}
\end{aligned}
$$

The first substage of the first round of the decryption process yields the following:

$$
\begin{aligned}
J_{11} &= Y_1 \odot U_1 \\
J_{12} &= Y_3 \boxplus U_2 \\
J_{13} &= Y_2 \boxplus U_3 \\
J_{14} &= Y_4 \odot U_4
\end{aligned}
$$

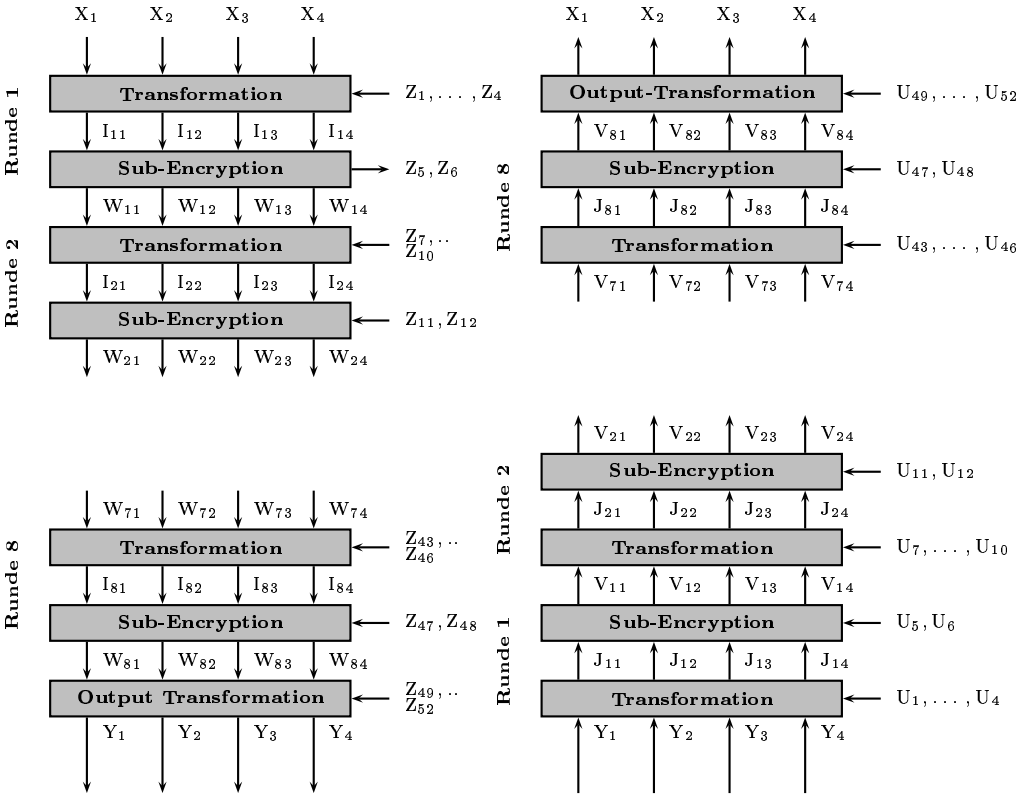


Abbildung 4.5: IDEA Ver- und Entschlüsselung

Thus substituting everything gives:

$$\begin{aligned}
J_{11} &= Y_1 \odot Z_{49}^{-1} = W_{81} \odot Z_{49} \odot Z_{49}^{-1} = W_{81} \\
J_{12} &= Y_2 \boxplus -Z_{50} = W_{82} \boxplus Z_{50} \boxplus -Z_{50} = W_{82} \\
J_{13} &= Y_3 \boxplus -Z_{51} = W_{83} \boxplus Z_{51} \boxplus -Z_{51} = W_{83} \\
J_{14} &= Y_4 \odot Z_{52}^{-1} = W_{84} \odot Z_{52} \odot Z_{52}^{-1} = W_{84}
\end{aligned}$$

Thus, we conclude that the output of the first substage of the decryption process is the same as the input to the last step in the encryption process.

Now consider the following relationships which can be derived from Figure 4.3:

$$\begin{aligned}
W_{81} &= I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
W_{82} &= I_{83} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
W_{83} &= I_{82} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
W_{84} &= I_{84} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})
\end{aligned}$$

where $MA_R(X, Y)$ is the right-hand output of the multiplication/addition structure with the two inputs X, Y and $MA_L(X, Y)$ is the left one.

Then we find:

$$\begin{aligned}
\mathrm{V}_{11} &= \mathrm{J}_{11} \oplus \mathrm{MA}_R(\mathrm{J}_{11} \oplus \mathrm{J}_{13}, \mathrm{J}_{12} \oplus \mathrm{J}_{14}) \\
&= \mathrm{W}_{81} \oplus \mathrm{MA}_R(\mathrm{W}_{81} \oplus \mathrm{W}_{82}, \mathrm{W}_{83} \oplus \mathrm{W}_{84}) \\
&= \underbrace{\mathrm{I}_{81} \oplus \mathrm{MA}_R(\mathrm{I}_{81} \oplus \mathrm{I}_{83}, \mathrm{I}_{82} \oplus \mathrm{I}_{84})}_{\mathrm{W}_{81}} \oplus \\
&\quad \mathrm{MA}_R\Big[\underbrace{\mathrm{I}_{81} \oplus \mathrm{MA}_R(\mathrm{I}_{81} \oplus \mathrm{I}_{83}, \mathrm{I}_{82} \oplus \mathrm{I}_{84})}_{\mathrm{W}_{81}} \\
&\qquad \oplus \underbrace{\mathrm{I}_{83} \oplus \mathrm{MA}_R(\mathrm{I}_{81} \oplus \mathrm{I}_{83}, \mathrm{I}_{82} \oplus \mathrm{I}_{84})}_{\mathrm{W}_{82}}, \\
&\qquad \underbrace{\mathrm{I}_{82} \oplus \mathrm{MA}_L(\mathrm{I}_{81} \oplus \mathrm{I}_{83}, \mathrm{I}_{82} \oplus \mathrm{I}_{84})}_{\mathrm{W}_{83}} \\
&\qquad \oplus \underbrace{\mathrm{I}_{84} \oplus \mathrm{MA}_L(\mathrm{I}_{81} \oplus \mathrm{I}_{83}, \mathrm{I}_{82} \oplus \mathrm{I}_{84})}_{\mathrm{W}_{84}}\Big] \\
&= \mathrm{I}_{81} \oplus \mathrm{MA}_R(\mathrm{I}_{81} \oplus \mathrm{I}_{83}, \mathrm{I}_{82} \oplus \mathrm{I}_{84}) \oplus \mathrm{MA}_R(\mathrm{I}_{81} \oplus \mathrm{I}_{83}, \mathrm{I}_{82} \oplus \mathrm{I}_{84}) \\
&= \mathrm{I}_{81}
\end{aligned}$$

Here we make several times use of the following property of the XOR-operation: Since

$$0 \oplus 0 = 0 \text{ and } 1 \oplus 1 = 0$$

we have in general

$$b_i \oplus b_i = 0 \text{ for any bit } b_i \in \{0, 1\}$$

Therefore, if a is an arbitrary bit-string then

$$a \oplus a = 0$$

and if z denotes another arbitrary bit string, obviously we get:

$$z \oplus (a \oplus a) = z \oplus 0 = z$$

In a similar fashion on can show:

$$\begin{aligned}
\mathrm{V}_{12} &= \mathrm{I}_{83} \\
\mathrm{V}_{13} &= \mathrm{I}_{82} \\
\mathrm{V}_{14} &= \mathrm{I}_{84}
\end{aligned}$$

Therefore, the output of the second substage of the decryption process is equal to the input of the next-to-last substage of the encryption process except for an interchange of the second and third blocks. It can be shown that this relationship holds at each corresponding pint in Figure 4.5 until

$$\begin{aligned}
\mathrm{V}_{81} &= \mathrm{I}_{11} \\
\mathrm{V}_{82} &= \mathrm{I}_{13} \\
\mathrm{V}_{83} &= \mathrm{I}_{12} \\
\mathrm{V}_{84} &= \mathrm{I}_{14}
\end{aligned}$$

Finally, because the output transformation of the decryption process is equal to the first substage of the encryption process except for an interchange of the second and third blocks, we have the output of the entire decryption process equal to the input of the encryption process.

4.2 Blowfish

Blowfish ist eine symmetrische Block Chiffre, die von **Bruce Schneier** und dessen Team bei Counterpane entwickelt wurde (siehe [193], pp. 336). Blowfish ist eine einfach strukturierte FEISTEL Chiffre mit variabler Schlüssellänge. Blowfish ist optimiert für schnelle Ausführung und einfache Implementierung. Daher hat Blowfish die folgenden Charakteristika und Design Prinzipien::

- ✘ Schnelligkeit
  Blowfish verschlüsselt Daten auf einen 32 Bit Mikroprozessor mit einer Rate von 26 Taktzyklen pro Byte.
- ✘ Kompaktheit
  Um Blowfish auszuführen werden lediglich 5 KB Speicherplatz benötigt.
- ✘ Einfachheit
  Das Blowfish Kryptosystem hat eine sehr einfach Struktur. Es werden nur sehr einfache Operationen eingesetzt: Addition, XOR Verknüpfungen und Zugriffe auf Tabellen. Dies hat den Vorteil, daß die Implementierung recht einfach ist und erleichtert die Aufgabe, die Stärke des Kryptosystems zu untersuchen.
- ✘ Variable Sicherheit
  Die Schlüssellänge der Chiffre ist variabel, es können bis zu 448 Bit lange Keys gewählt werden. Diese Flexibilität erlaubt einen Kompromiss zwischen hoher Verarbeitungsgeschwindigkeit und hoher Sicherheit.

Abbildung 4.6: Bruce Schneier.

Blowfish verschlüsselt 64-Bit Klartextblöcke in 64-Bit Chiffreblöcke. Das Blowfish Kryptosystem ist in einer Reihe von Softwareprodukten implementiert. Bisher ist die Sicherheit dieses Verfahrens unbestritten.

4.2.1 Erzeugung der Rundenschlüssel und S-Boxen

Das Blowfish Kryptosystem benutzt einen Schlüssel mit variabler Länge von 32 bis 448 Bit. Werden – wie üblich – die 32 Bit als ein *Word* bezeichnet, benutzt Blowfish also Schlüssel der Länge 1 bis 14 Words. Dieser Inputschlüssel

wird dazu verwendet, eine Vielzahl von Größen zu generieren, die im Algorithmus verwendet werden. Diese Parametermüssen berechnet werden, bevor eine Ver– bzw. Entschlüsselung von Nutzdaten durchgeführt werden kann. Aus den Schlüssel werden abgeleitet:

- 18 Rundenschlüssel mit einer Länge von je 32 Bit
- 8×32 **S-Boxes**, diese enthalten insgesamt 1.024 32-Bit Einträge.

Die primären Schlüssel werden in einem sogenannten **K-Array** gespeichert:

$$K_1, K_2, \ldots K_j; \qquad 1 \leq j \leq 14.$$

Die 18 Teilschlüssel werden in einem sogenannten **P-Array** hinterlegt:

$$P_1, P_2, \ldots P_{18};$$

Zusätzlich werden vier S-Boxen generiert, jede S-Box besteht aus 256 Einträgen der Länge von 32-Bit:

$$\begin{aligned} &S_{1,0}, S_{1,1}, \ldots, S_{1,255} \\ &S_{2,0}, S_{2,1}, \ldots, S_{2,255} \\ &S_{3,0}, S_{3,1}, \ldots, S_{3,255} \\ &S_{4,0}, S_{4,1}, \ldots, S_{4,255} \end{aligned}$$

Bei der Erzeugung des P-Arrays und der S-Boxen werden folgende Schritte ausgeführt:

❶ Zunächst wird der P-Array und anschließend werden die vier S-Boxen der Reihe nach initialisiert. Dabei werden die Nachkommastellen der Konstanten π verwendet. Die ersten 32 Bit der Nachkommastellen von π werden der Variablen P_1 zugeordnet, usw. In Hexadezimalschreibweise sind diese Variablen dann folgendermaßen belegt:

$$\begin{aligned} P_1 &= \texttt{243F6A88} \\ P_2 &= \texttt{85A308D3} \\ &\vdots \quad \vdots \\ S_{4,254} &= \texttt{578FDFE3} \\ S_{4,255} &= \texttt{3AC372E6} \end{aligned}$$

❷ Der zweite Schritt besteht darin, den n Schritt 1 initialisierten P-Array und den K-Array bitweise zu XORen. Falls erforderlich werden dabei K-Array Words mehrfach verwendet. Betrachten wir den Fall eines Schlüssels

K mit maximaler Länge von 14 Words (je 32 Bit), dann folgt:

$$\begin{aligned}
P_1 &= P_1 \oplus K_1 \\
P_2 &= P_2 \oplus K_2 \\
&\vdots \quad \vdots \\
P_{14} &= P_{14} \oplus K_{14} \\
P_{15} &= P_{15} \oplus K_1 \\
P_{16} &= P_{16} \oplus K_2 \\
P_{17} &= P_{17} \oplus K_3 \\
P_{18} &= P_{18} \oplus K_4
\end{aligned}$$

❸ Im nächsten Schritt wird der 64 Bit Block, bestehend aus 64 0en mit dem Blowfish Algorithmus verschlüsselt. Dabei werden die aktuellen Werte der P– und S Arrays benutzt. Das Resultat dieser Verarbeeitung ist ein 64 Bit Block, der in zwei Blöcke der Länge von 32 Bit aufgeteilt wird. P_1 und P_2 werden durch diese beiden Blöcke ersetzt.

❹ Der Output des dritten Schritts wird mit dem Blowfish Algorithmus und den aktuellen P- und S-Arrays verschlüsselt. Der resultierende Chiffretext ersetzt die P-Aary Einträge P_3 und P_4.

❺ Dieses verfahren wird fortgeführt, bis sämtliche Elemente des P-Arrays der Reihe nach aktualisiert sind. Anschließend werden die Einträge der S-Boxen auf die gleiche Weise neu belegt. Bei jedem Schritt wird die Ausgabe des sich stetig ändernden Blowfish Algorithmus verwendet.

Der Update-Prozess der S-Boxen und des P-Arrays kann folgendermaßen zusammengefaßt werden:

$$\begin{aligned}
P_1, P_2 &= \mathrm{E}_{P,S}[0] \\
P_3, P_4 &= \mathrm{E}_{P,S}[P_1 \,||\, P_2] \\
P_5, P_6 &= \mathrm{E}_{P,S}[P_3 \,||\, P_4] \\
&\vdots \quad \vdots \\
P_{15}, P_{16} &= \mathrm{E}_{P,S}[P_{13} \,||\, P_{14}] \\
P_{17}, P_{18} &= \mathrm{E}_{P,S}[P_{15} \,||\, P_{16}] \\
S_0, S_1 &= \mathrm{E}_{P,S}[P_{17} \,||\, P_{18}] \\
&\vdots \quad \vdots \\
S_{4,254}, S_{4,255} &= \mathrm{E}_{P,S}[S_{4,252} \,||\, S_{4,253}]
\end{aligned}$$

Dabei bezeichnet $\mathrm{E}_{P,S}[Y]$ den Chiffretext, der entsteht, wenn Y durch den Blowfish Algorithmus mit den Arrays S und P verschlüsselt wird.

Es sind insgesamt 521 Verarbeitungsläufe des kompletten Blowfish Algorithmus nötig, um die P- und S-Arrays mit den Endzuständen zu erstellen. Dazu benötigt man:

$\longrightarrow$ 9 Ausführungen, um die Teilschlüsselpaare P_i, P_j zu erzeugen

$\longrightarrow$ 512 Ausführungen, um die Paare von S-Box Einträgen S_i, S_j zu generieren.

Aus diesem Grund ist der Blowfish Algorithmus nicht für kryptographische Anwendungen geeignet, die eine häufigen Änderung des Schlüssels K erfordern. Um die Performanz dieses Algorithmus zu erhöhen, können die P- und S-Arrays gespeichert werden, anstatt bei jedem Gebrauch des Blowfish Algorithmus eine Ableitung dieser Werte aus dem Schlüssel vorzunehmen. Dies erfordert mehr als 4 Kbyte Speicherplatz. Daher ist Blowfish nicht für Anwendungen geeignet, die über einen beschränkten Speicherplatz verfügen wie beispielsweise Smartcards.

4.2.2 Ver- und Entschlüsselung

Bruce Schneiers allgemeines Design Prinzip des Blowfish Algorithmus von Anfang an ist die Verwendung von einfachen Operationen. Blowfish operiert mit:

- ☞ Addition: Die Addition von Words – diese Operation wird mit '+' bezeichnet – wird modulo 2^{32} ausgeführt.
- ☞ Bitweise XOR Verknüpfung: Diese Operation wird mit $\oplus$ bezeichnet.

Die für die Kryptographie wichtige Eigenschaft dieser beiden Opertaon ist, daß sie nicht kommutieren.

Die Abbildung [4.7] zeigt das Schema der Verschlüsselung mit der Blowfish Chiffre. Der Klartextblock wird in zwei 32 Bit Blockhälften LE_0 and RE_0 aufgespaltet. Die Variablen LE_i und RE_i stehen für die linke bzw. rechte Datenblockhälfte, nachdem die $i$te Verarbeitungsrunde beendet wurde. Insgesamt werden 16 Runden durchlaufen.

Der Blowfish Verschlüsselungsalgorithmus kann durch den folgenden Pseudocode definiert werden:

$$
\begin{aligned}
&\text{for } i = 1 \text{ to } 16 \text{ do} \\
&\qquad \mathrm{RE}_i = \mathrm{LE}_{i-1} \oplus \mathrm{P}_i \ ; \\
&\qquad \mathrm{LE}_i = \mathrm{F}[\mathrm{RE}_i] \oplus \mathrm{RE}_{i-1} \ ; \\
&\mathrm{LE}_{17} = \mathrm{RE}_{16} \oplus \mathrm{P}_{18} \ ; \\
&\mathrm{RE}_{17} = \mathrm{LE}_{16} \oplus \mathrm{P}_{17} \ ;
\end{aligned}
$$

Der resultierende Chiffretext ist in den beiden Variablen LE_{17} und RE_{17} enthalten. Eine zentrale Rolle im Blowfish Algorithmus spielt die Rundenfunktion F. Die Details dieser Funktion sind in der Abbildung [4.9] skizziert. Der Input ist ein 32-Bit Block, der in vier 8-Bit Blöcke aufgeteilt wird. Diese vier Bytes werden mit a, b, c und d bezeichnet.

Mit diesen vier Variablen kann die F-Funktion geschrieben werden in der Form:

$$\mathrm{F}(a, b, c, d) = ((\mathrm{S}_{1,a} + \mathrm{S}_{2,b}) \ \oplus \ \mathrm{S}_{3,c}) + \mathrm{S}_{4,d}$$

Daher werden in jeder Runde die folgenden Operationen ausgeführt:

$\longrightarrow$ Addition modulo 2^{32}

$\longrightarrow$ XOR

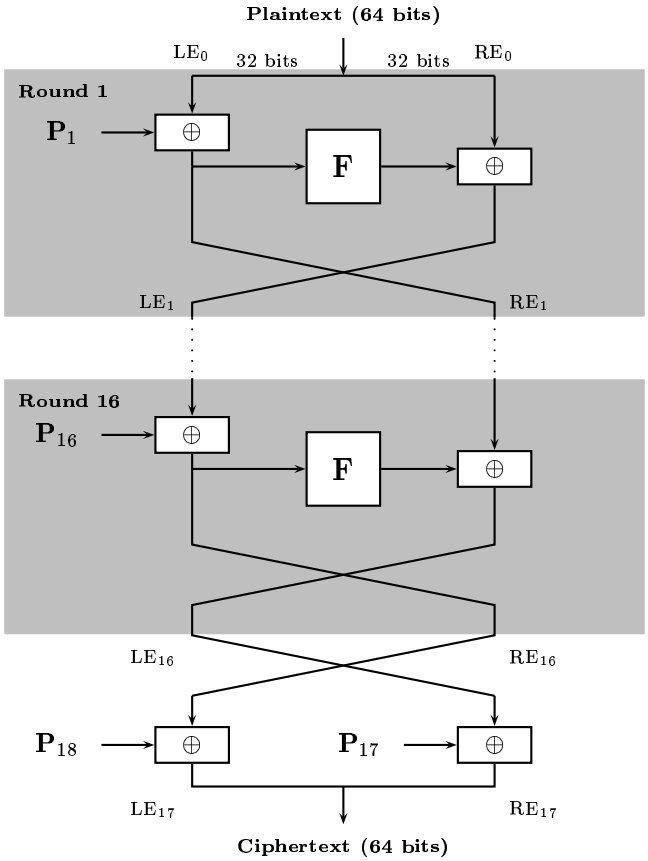


Abbildung 4.7: Verschlüsselung mit dem Blowfish Kryptosystem.

$\longrightarrow$ Substitution durch S-Boxen.

Der Ablauf der Dechiffrierung mit dem Blowfish Algorithmus ist in der Abbildung [4.8] dargestellt. It is easily derived from the encryption algorithm. In the decryption case, the 64 bits of ciphertext are initially assigned to the two 32 bit variables LD_0 and RD_0. In this algorithm we denote by LD_i and RD_i the left and right halves of the data after round i. As with most ciphers, Blowfish decryption uses the subkeys in revers order. However, unlike most block ciphers, Blowfish decryption occurs in the same algorithmic order as encryption. Thus the decryption algorithm is:

$$
\begin{aligned}
&\text{for } i = 1 \text{ to } 16 \text{ do}\\
&\qquad \mathrm{RD}_i = \mathrm{LD}_{i-1} \oplus \mathrm{P}_{19-i}\ ;\\
&\qquad \mathrm{LD}_i = \mathrm{F}[\mathrm{RD}_i] \oplus \mathrm{RD}_{i-1}\ ;\\
&\mathrm{LD}_{17} = \mathrm{RD}_{16} \oplus \mathrm{P}_1\ ;\\
&\mathrm{RD}_{17} = \mathrm{LD}_{16} \oplus \mathrm{P}_2\ ;
\end{aligned}
$$

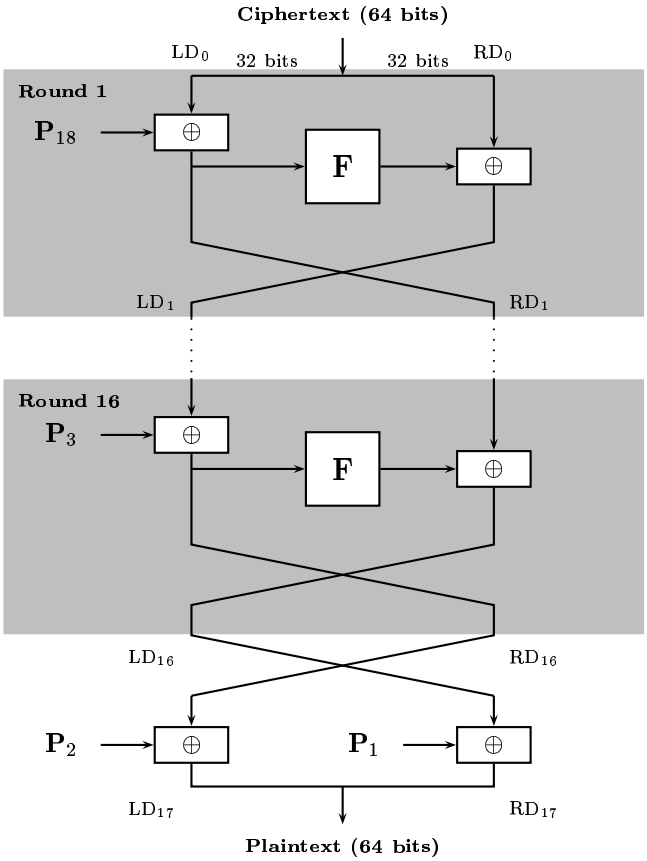


Abbildung 4.8: Blowfish Entschlüsselung.

4.2.3 Zusammenfassung

Das Design des Blowfish Algorithmus zeigt eine Reihe interessanter Eigenschaften:

- ☞ Die S-Boxen sind abh.ängig vom Schlüssel.
- ☞ Die Rundenschlüssel und die S-Boxen werden durch einen Prozess erstellt, der den Blowfish Algorithmus selbst mehrfach anwendet. Dadurch wird die Kryptoanalyse wiederum sehr schwierig.
- ☞ In jeder Runde werden Operationen auf beide Hälften der Datenblöcke angewendet. Dies ist eine Verbesserung zu der klassischen FEISTEL Chiffre, bei der pro Runde nur eine Datenblockhälfte verarbeitet wird. Diese Eigenschaft erhöht die kryptographische Stärke des Kryptosystems.
- ☞ Hinsichtlich Brute Force Attacken ist der Blowfish Algorithmus durch eine geeignete Wahl der Schlüssellänge nicht zu brechen. Die Schlüssellänge kann bis 448 Bits gewählt werden.

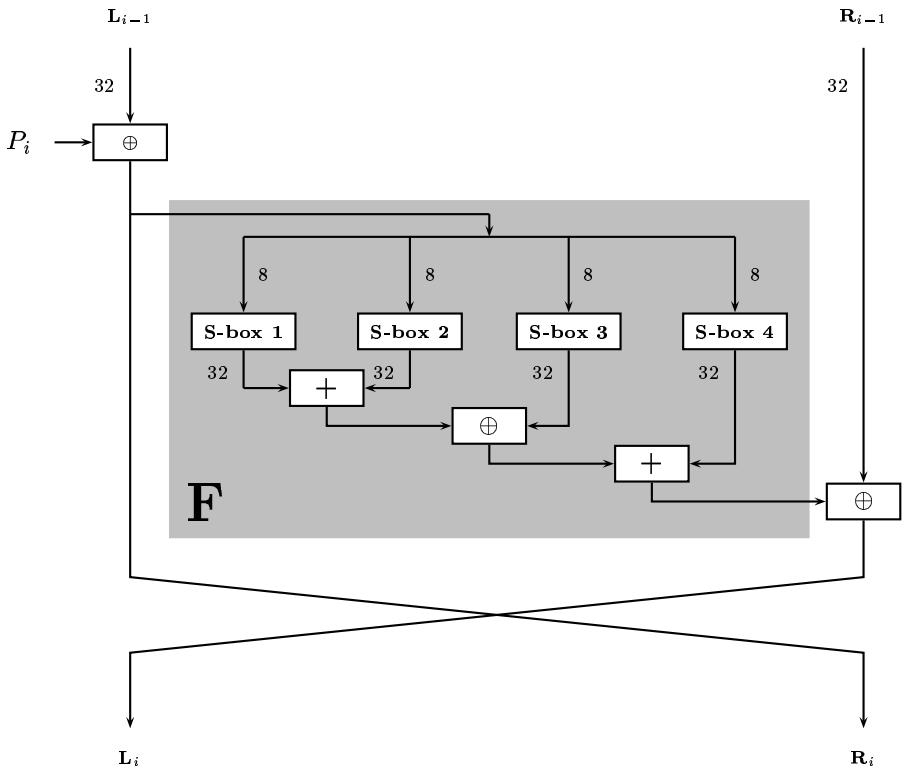


Abbildung 4.9: Details der Blowfish F-Funktion.

☞ Der Blowfish Algorithmus ist sehr effektiv und schnell während der Ausführung. Auf einem Pentium benötigt die Ausführung einer Runde beeindruckend wenig Taktzyklen im Vergleich zu anderen Verschlüsselungsverfahren.

4.3 RC4 und WEP

4.3.1 Die RC4 Stromchiffre

Die RC4 Chiffre [3] ist eine Strom Chiffre mit variabler Schlüssellänge. RC4 wurde 1987 von RON RIVEST für die RSA Data Security, heute RSA Security, entwickelt. RC4 wurde als proprietäre Chiffre für RSA entwickelt, daher waren über Jahre hinweg die Details des Algorithmus nur verfügbar und zugänglich nach Unterzeichnung einer Vertraulichkeitsvereinbarung.

Im September 1994 veröffentlichte jemand anonym den Quellcode der RC4 Chiffre in der Mailing Liste *Cyperpunks*[4]. Der Code erreichte schnell die Usenet newsgroup `sci.crypt` und über das Internet zu vielen ftp-Servern auf der ganzen Welt. Leser mit legalen Versionen von RC4 bestätigten die Kompatibilität der gehackten Version mit der proprietären Version von RSA Security. Die RSA Data Security versuchte, den Schaden zu begrenzen, indem sie darauf bestand, daß der Quellcode nach wie vor Betriebsgeheimnis ist, selbst wenn er publiziert ist – doch es war zu spät. Seitdem wurde die Chiffre im Usenet diskutiert und analysiert, auf Konferenzen vorgestellt und in Kryptovorlesungen vorgestellt (sogar an der BA Mannheim).

RC4 ist eine sehr einfache Chiffre. Der Schlüsselstrom ist unabhängig vom Klartext. Die zentrale Komponente von RC4 sind 256 Variable

$$s_0, s_1, \ldots, s_{255},$$

die *S-Box* heißen. Die Größen s_i sind eine Permutation der Zahlen von 0 bis 255; diese Permutation ist eine Funktion des Schlüssels, der eine variable Länge hat. Weiterhin werden in der Chiffre zwei Zähler i und j benutzt, die auf 0 initialisiert sind.

Ein zufälliges Byte B des Schlüsselstroms wird durch folgende Schritte generiert:

① Berechne

$$i = (i + 1) \bmod 256$$

② Berechne

$$j = (j + s_i) \bmod 256$$

③ Vertausche s_i und s_j

④ Berechne

$$t = (s_i + s_j) \bmod 256$$

[3] Eine ergiebige Web Site über den RC4 Algorithmus ist unter folgender URL zu finden:

http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html

[4] Die Original Mail kann unter der URL

http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0

eingesehen werden.

⑤ Setze

$$B = s_t$$

Das Byte B wird anschließend mit dem Klartext XOR-verknüpft um den Chiffretext zu erhalten oder – auf Empfängerseite – mit dem Chiffretext geXORt, um den Klartext zurückzugewinnen. Die Verschlüsselung ist sehr effektiv und schnell, etwa den Faktor 10 schneller als DES.

Die Initialisierung der S-Box ist ebenfalls einfach. Dabei werden folgende Schritte durchlaufen:

① Beginne mit

$$s_0 = 0; s_1 = 1; \ldots ; s_{255} = 255$$

② Ein weiterer 256-Byte Array wird mit dem Schlüssel K gefüllt, falls erforderlich wird der Schlüssel wiederholt, um den kompletten Array zu füllen:

$$K_0, K_1, \ldots , K_{255}$$

③ Setze den Index $j = 0$. Dann führe folgende Schleife aus:

```
for(i=0; i<256;i++)
 {
   j=(j+s[i]+K[i]) mod 256
   vertausche s[i] und s[j]
 }
```

Das ist alles.

RSA Data Security behauptet, daß der Algorithmus resistent gegen differentieller als auch linearer Kryptoanalyse ist. Weiterhin scheint das Verfahren keine kleine Zyklen zu haben und ist hochgradig nichtlinear. RC4 kann $256! \times 256^2 \approx 2^{1700}$ verschiedene Zustände annehmen.

Beispiel:

Um das RC4 Verfahren zu verstehen, kann man das Verfahren folgendermaßen durchspielen. Wir wählen eine S-Box– und Schlüssellänge von 8 Bytes, i.e.

$$S = [s[0], s[1], \ldots , s[7]]$$
$$K = [k[0], k[1], \ldots , k[7]]$$

Den Key selbst setzen wir auf:

$$K = [k[0], k[1], k[2], k[3], k[4], k[5], k[6], k[7]] = [1, 4, 3, 2, 7, 6, 0, 5]$$

In einem ersten Schritt wird die S-Box initialisiert. Dazu wird mit der Belegung

$$S = [s[0], s[1], s[2], s[3], s[4], s[5], s[6], s[7]] = [0, 1, 2, 3, 4, 5, 6, 7]$$

gestartet. Anschließend wird der Zähler j auf den Wert $j = 0$ gesetzt und die Iteration

```
for(i=0; i<8;i++)
 {
   j=(j+s[i]+k[i]) mod 8
   vertausche s[i] und s[j]
 }
```

durchlaufen. Dies führen wir jetzt Schritt für Schritt durch.

- Step 1: $i = 0$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (0 + s[0] + k[0]) \bmod 8 \\ &= 1 \end{aligned}$$

Vertausche $s[0], s[1]$, dann erhalten wir nach dem ersten Schritt:

$$S = \big[1, 0, 2, 3, 4, 5, 6, 7\big]$$

- Step 2: $i = 1$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (1 + s[1] + k[1]) \bmod 8 \\ &= (1 + 0 + 4) \bmod 8 \\ &= 5 \end{aligned}$$

Vertausche $s[1], s[5]$, dann erhalten wir nach dem zweiten Schritt:

$$S = \big[1, 5, 2, 3, 4, 0, 6, 7\big]$$

- Step 3: $i = 2$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (5 + s[2] + k[2]) \bmod 8 \\ &= (5 + 2 + 3) \bmod 8 \\ &= 2 \end{aligned}$$

Vertausche $s[2], s[2]$, dann erhalten wir nach dem dritten Schritt:

$$S = \big[1, 5, 2, 3, 4, 0, 6, 7\big]$$

- Step 4: $i = 3$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (2 + s[3] + k[3]) \bmod 8 \\ &= (2 + 3 + 2) \bmod 8 \\ &= 7 \end{aligned}$$

Vertausche $s[3], s[7]$, dann erhalten wir nach dem vierten Schritt:

$$S = \big[1, 5, 2, 7, 4, 0, 6, 3\big]$$

- Step 5: $i = 4$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (7 + s[4] + k[4]) \bmod 8 \\ &= (7 + 4 + 7) \bmod 8 \\ &= 2 \end{aligned}$$

Vertausche $s[4], s[2]$, dann erhalten wir nach dem fünften Schritt:

$$S = \left[1, 5, 4, 7, 2, 0, 6, 3\right]$$

- Step 6: $i = 5$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (2 + s[5] + k[5]) \bmod 8 \\ &= (2 + 0 + 6) \bmod 8 \\ &= 0 \end{aligned}$$

Vertausche $s[5], s[0]$, dann erhalten wir nach dem sechsten Schritt:

$$S = \left[0, 5, 4, 7, 2, 1, 6, 3\right]$$

- Step 7: $i = 6$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (0 + s[6] + k[6]) \bmod 8 \\ &= (0 + 6 + 0) \bmod 8 \\ &= 6 \end{aligned}$$

Vertausche $s[6], s[6]$, dann erhalten wir nach dem siebten Schritt:

$$S = \left[0, 5, 4, 7, 2, 1, 6, 3\right]$$

- Step 8: $i = 7$

$$\begin{aligned} j &\longleftarrow (j + s[i] + k[i]) \bmod 8 \\ &= (6 + s[7] + k[7]) \bmod 8 \\ &= (6 + 3 + 5) \bmod 8 \\ &= 6 \end{aligned}$$

Vertausche $s[7], s[6]$, dann erhalten wir nach dem achten Schritt:

$$S = \left[0, 5, 4, 7, 2, 1, 3, 6\right]$$

Damit ist die S-Box initialisiert und (schlüsselabhängig) auf den Wert

$$S = \left[0, 5, 4, 7, 2, 1, 3, 6\right]$$

gesetzt.

Mit dieser S-Box können die Schlüsselbytes folgendermaßen erzeugt werden:

Ein zufälliges Byte B des Schlüsselstroms wird durch folgende Schritte generiert:

① Es werden zwei Zähler i und j benutzt mit der Initialisierung $i = j = 0$

② Berechne

$$i = (i + 1) \bmod 8$$

③ Berechne

$$j = (j + s[i]) \bmod 8$$

④ Vertausche $s[i]$ und $s[j]$

⑤ Berechne

$$t = (s[i] + s[j]) \bmod 8$$

⑥ Setze

$$B = s[t]$$

1. Byte 1:

   $$i \longleftarrow (i + 1) \bmod 8 = 1$$

   damit wird der Index j zu:

   $$j \longleftarrow (j + s[i]) \bmod 8 = s[1] \bmod 8 = 5$$

   Vertauschen von $s[i]$ und $s[j]$, i.e. $s[1]$ und $s[5]$ führt auf:

   $$S = \big[0, 1, 4, 7, 2, 5, 3, 6\big]$$

   und

   $$t \leftarrow (s[1] + s[5]) \bmod 8 = (1 + 5) \bmod 8 = 6$$

   Damit:

   $$B_1 = s[t] = s[6] = 3.$$

   Verwendet man den RC4 Algorithmus als Pseudozufallszahlengenerator um eine (pseudeo)zufällige Bitfolge zu generierem, kann hier die Parität genutzt werden, z. B.

   $$P(B_1) = P(s[t]) = P(s[6]) = P(3) \longrightarrow b_1 = 1.$$

2. Byte 2:

   $$i \longleftarrow (1 + 1) \bmod 8 = 2$$

   damit wird der Index j zu:

   $$j \longleftarrow (j + s[i]) \bmod 8 = (5 + s[2]) \bmod 8 = (5 + 4) \bmod 8 = 1$$

Vertauschen von $s[i]$ und $s[j]$, i.e. $s[2]$ und $s[1]$ führt auf:

$$S = [0, 4, 1, 7, 2, 5, 3, 6]$$

und

$$t \leftarrow (s[2] + s[1]) \bmod 8 = (1 + 4) \bmod 8 = 5$$

Damit:

$$B_2 = s[t] = s[5] = 5$$

und

$$P(B_2) = P(s[t]) = P(s[5]) = P(5) \longrightarrow b_2 = \texttt{1}.$$

3. Byte 3:

$$i \longleftarrow (i + 1) \bmod 8 = 3$$

damit wird der Index j zu:

$$j \longleftarrow (j + s[3]) \bmod 8 = (1 + s[3]) \bmod 8 = (1 + 7) \bmod 8 = 0$$

Vertauschen von $s[i]$ und $s[j]$, i.e. $s[3]$ und $s[0]$ führt auf:

$$S = [7, 4, 1, 0, 2, 5, 3, 6]$$

und

$$t \leftarrow (s[0] + s[3]) \bmod 8 = (7 + 0) \bmod 8 = 7$$

Damit:

$$B_3 = s[t] = s[7] = 6$$

und

$$P(B_3) = P(s[t]) = P(s[7]) = P(6) \longrightarrow b_3 = \texttt{0}.$$

4. Byte 4:

$$i \longleftarrow (i + 1) \bmod 8 = 4$$

damit wird der Index j zu:

$$j \longleftarrow (j + s[i]) \bmod 8 = (0 + s[4]) \bmod 8 = (0 + 2) \bmod 8 = 2$$

Vertauschen von $s[i]$ und $s[j]$, i.e. $s[4]$ und $s[2]$ führt auf die neue S-Box:

$$S = [7, 4, 2, 0, 1, 5, 3, 6]$$

und

$$t \leftarrow (s[4] + s[2]) \bmod 8 = (4 + 2) \bmod 8 = 6$$

Damit:

$$B_4 = s[t] = s[6] = 3$$

und

$$P(B_4) = P(s[t]) = P(s[6]) = P(3) \longrightarrow b_4 = \texttt{1}.$$

Führt man diese Berechnung fort, ergibt sich Bytefolge:

$$3, 5, 6, 0, 0, 6, 3, 4, 5, 0, 1, 0, 5, 6, 6, 5, 6, 1, 0, 7, 1, 5, 3, 0, 5, 4, 6, 6, 2, ...$$

bzw. die Bitfolge

```
1,1,0,0,0,0,1,0,1,0,1,0,1,0,0,1,0,1,0,1,1,1,1,1,0,1,0,0,0,0,...
```

4.3.2 WEP

Das bekannteste Einsatzgebiet des RC4 Algorithmus ist die in der IEEE 802.11 Spezifikation standardisierte WEP Verschlüsselung. Zunächst sei die Zielsetzung des *wired equivalent privacy* Verfahrens (WEP) zitiert, die im Standarddokument formuliert wurde:

> *Eavesdropping is a familiar problem to users of other types of wireless technology. IEEE 802.11 specifies a wired LAN equivalent data confidentiality algorithm. Wired equivalent privacy is defined as protecting authorized users of a wireless LAN from casual eavesdropping. This service is intended to provide functionality for the wireless LAN equivalent to that provided by the physical security attributes inherent to a wired medium.*

Wie in der Zielsetzung klar formuliert wurde, soll WEP eine Sicherheit der Datenübertragung via Funkwellen gewährleisten, die gleichwertig zu der eines drahtgebundenen Netzes ist. Der genaue Wortlaut des Standards ist hier sehr wichtig: Die Entwickler hatten *nicht* die Absicht, eine Sicherheit zu gewährleisten, die über der eines Ethernets hinausgeht. Allein schon der Name des Algorithmus *wired equivalent privacy* deutet auf die Absicht der Entwickler hin. Wie die Praxis jedoch gezeigt hat, ist ein zu einem drahtgebundenen Netz gleichwertiger Sicherheitslevel bei drahtlosen Netzen nicht ausreichend. Es ist letztendlich die Annahme, daß drahtlose Netze so sicher sein müssen wie drahtgebundene Netze, die falsch ist. Andere Schwachstellen wie die Wahl eines CRC-32 Verfahrens anstatt des Message Digest Algorithmus MD-5, oder eines anderen sicheren Hash Verfahrens, verschlimmern das Problem nur.

4.3.3 Funktionsweise von WEP

WEP benutzt einen geheimen Schlüssel K, der zwischen den 802.11 Knoten ausgetauscht wird[5], um die Datenframes (ISO/OSI Layer 2) zu verschlüsseln. Der WEP Standard sieht die Bildung einer CRC-32 Prüfsumme vor, um Datenintegrität zu gewährleiten. Diese Prüsumme wird ebenfalls mit dem symmetrischen Verfahren mit Hilfe des geheimen Schlüssels verschlüsselt. Die Entschlüsselung ist der inverse Prozess zur Verschlüsselung. Der empfangene Frame wird mit Hilfe des geheimen Schlüssels dechiffriert, der Empfänger berechnet die Prüfsumme über den Klartext erneut und wird mit der erhaltenen Prüfsumme abgeglichen.

Das im WEP verwendete Chiffrierverfahren ist der RC4 Algorithmus, der hier dazu verwendet wird, einen *One Time Pad* zu erzeugen, i.e. einen Keystrom

[5] Also beispielsweise zwischen dem Access Point und den mobilen Stationen.

beliebiger Länge für genau eine Übertragung. Die Schlüssellänge im WEP Protokoll reicht – je nach Implementierung – von 40 bis 128 Bits. Ein sogenannter Initialisierungsvektor IV, der während des Verschlüsselungsprozesses benötigt wird, hat aber nur die Länge von 24 Bit. Das WEP Protokoll – also der 802.11i Standard – sieht *keinerlei* Mechanismen vor, wie der geheime Schlüssel zwischen den Knoten ausgetauscht wird.

WEP benutzt also einen geheimen Schlüssel K, dieses Geheimnis kennen nur die kommunizierenden Parteien in dem WLAN. Dieser wird genutzt, den Datenframe zu verschlüsseln.

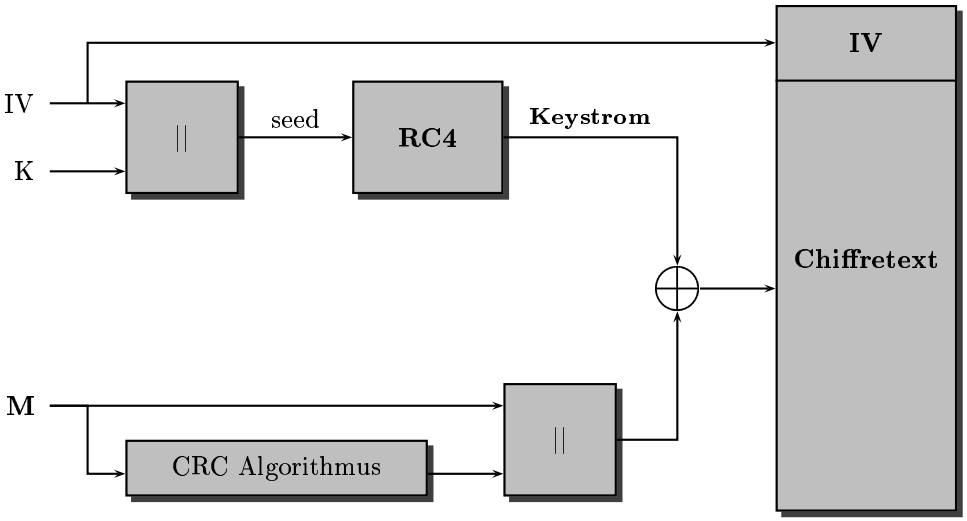


Abbildung 4.10: Verschlüsselung eines WEP Frames.

Die Verschlüsselung eines Frames geschieht in folgenden Schritten:

1. **Integritätsprüfsumme**
   Zunächst wird eine Integritätsprüfsumme $c(M)$ der Message M berechnet. Dies geschieht mittels CRC-32 Cyclic Redundancy Code. Die Nachricht und die Checksumme werden konkateniert ($||$) zu dem Klartext

   $$P =< M, c(M) >$$

   der als Input des zweiten Schrittes dient. Weder $c(M)$ noch P hängen vom gewählten Schlüssel ab.

2. **Verschlüsselung**
   Im zweiten Schritt wird der Klartext P mit der RC4 Chiffre verschlüsselt. Dabei wird ein Initialisierungsvektor (IV) v benutzt. Der RC4 Algorithmus erzeugt einen Schlüsselstrom, i.e. eine beliebig lange Folge pseudozufälliger Bytes – als Funktion des Initialisierungsvektors v und des Schlüssels K. Diesen Schlüsselstrom bezeichnen wir mit RC4(v, K). Dann

wird der Klartext P mit dem Schlüsselstrom geXORt ($\oplus$). Aus dieser Operation folgt der Chiffretext C:

$$C = P \oplus \text{RC4}(v, K)$$

3. **Übertragung**
   Schließlich wird der Initialisierungsvektor IV im Klartext und der Chiffretext übertragen. Der Empfänger benötigt zum Dechiffrieren des Datenblocks den Initial Vector IV.

Ein Empfänger muß das durch WEP verschlüsselte Frame dechiffrieren, indem er einfach den Verschlüsselungsvorgang umkehrt. Da der verwendete Initialisierungsvektor mit dem Chiffretext übertragen wird, generiert der Empfänger – er kennt natürlich auch den geheimen Schlüssel K – den Schlüsselstrom $\text{RC4}(v, K)$. Der empfangene Chiffretext C wird nun mit $\text{RC4}(v, K)$ geXORt, so daß der Klartext P erhalten wird

$$\begin{aligned} P' &= C \oplus \text{RC4}(v, K) \\ &= \big(P \oplus \text{RC4}(v, K)\big) \oplus \text{RC4}(v, k) \\ &= P \end{aligned}$$

In einem nächsten Schritt verifiziert der Empfänger die Prüfsumme des entschlüsselten Klartexts P', indem dieser in die Form

$$< M', c' >$$

aufgesplittet wird. Die Prüfsumme $c(M')$ wird berechnet. Wenn dieser Wert mit c' übereinstimmt, ist die Integrität gewährleistet, i.e. die Daten wurden während der Übertragung nicht geändert. i

4.4 Der RC5 Algorithmus

Der Krypto-Guru **Ron Rivest** entwickelte 1994 die symmetrische Blockchiffre **RC5**, welche folgende Design-Features aufweist:

- **Geeignet für Hardware – und Softwareimplementierungen**
  Die RC5 Chiffre verwendet sehr einfache Operationen, die von direkt von Mikroprozessoren umgesetzt werden können.
- **Schnelligkeit**
  RC5 ist ein einfacher Algorithmus, der auf Words[6] basiert. Die grundlegenden Operationen arbeiten auf kompletten Datenworten.
- **Portabilität**
  Der RC5 Algorithmus kann auf unterschiedliche Word-Längen – entsprechend der Prozessorplattform – angepasst werden. Die wird dadurch erreicht, daß die Länge eines Worts durch die Anzahl der Bits, die in ein Word passen, parametrisiert wird. Unterschiedliche Word-Längen führen auf unterschiedliche Algorithmen.
- **Variable Anzahl von Runden**
  Die Anzahl der Runden ist ein zweiter Parameter der RC5 Chiffre. Dieser Parameter erlaubt einen Kompromiss zwischen höherer Verarbeitungsgeschwindigkeit einerseits und höherer Sicherheit andererseits.
- **Variable Schlüssellänge**
  Die Länge des Schlüssels ist ein weiterer Parameter der RC5 Chiffre. Die Wahl der Schlüssellänge erlaubt einen weiteren Kompromiss zwischen Performance und Sicherheit.
- **Einfachheit**
  Aufgrund seiner einfachen Struktur ist RC5 einfach zu implementieren. Weiterhein erleichtert dies die Aufgabe, die Stärke des Verfahrens einzuschätzen.
- **Niedrige Speicheranforderungen**
  RC5 erfordert wenig Speicherplatz. Dadurch ist RC5 für Anwendungen auf Smartcards und anderen Geräten mit eingeschränkten Speicherplatz geeignet.
- **Hoher Grad an Sicherheit**
  Mit der geeigneten Wahl der Parameter bietet RC5 einen hohen Grad an Sicherheit.
- **Datenabhängige Rotationen**
  Die RC5 Chiffre beinhaltet sogenannte Rotationen. Dies sind zirkuläre Bitshifts, wobei die Anzahl der verschobenen Bitstellen datenabhängig ist. Dieses Feature scheint den Algorithmus gegenüber Angriffen zu stärken.

Der RC5 Algorithmus ist patentiert von RSA Data Security, heute RSA Security.

[6] In diesem Kontext steht der Begriff *Word* für die Blockgröße in Bit, die die CPU bei der Verarbeitung in einem Schritt abarbeiten kann; ein typischer Wert ist 32 Bits.

4.4.1 RC5 Parameter

Wie oben erwähnt, erlaubt RC5 das Setzen dreier freier Parameter:

Parameter	Definition	Werte
w	Word-Länge in Bit; RC5 verschlüssel 2Word Blöcke	16, 32, 64
r	Anzahl der Runden	$0, 1, \ldots, 255$
b	Anzahl von 8-bit Bytes (Oktette) im Schüssel K	$0, 1, \ldots, 255$

Mit Hilfe dieser Parameter kann mit RC5 ein Klartextblock der Länge 32, 64 oder 128 Bit in einen Chiffreblock der gleichen Länge abgebildet werden. Die Schlüssellänge reicht von 0 bis 2.040 Bits ($= 8 \times 255$). Eine spezielle Wahl der Parameter wird mit RC5-$w/r/b$ bezeichnet. Das bedeutet, RC5-32/12/16 bearbeitet 32 Bit words (i.e. 64 Bit Klartextblöcke und 64 Bit Chiffretextblöcke), 12 Runden im Ver– bzw. Entschlüsselungsalgorithmus und eine Schlüssellänge von 16 Bytes (=128 Bits).

4.4.2 RC5 Schlüsselerzeugung

Der RC5 Verschlüsselungsalgorithmus benutzt einen Satz sehr komplexer Operationen um aus dem Schlüssel K insgesamt t Rundenschlüssel abzuleiten. Zwei Subkeys werden in jeder Runde benötigt und zwei Subkeys werden für eine zusätzliche Operation benötigt, die zu keiner Runde gehören. Dies impliziert: $t = 2r + 2$. Jeder dieser Subkeys hat eine Länge von w Bits, i.e. 1 Word.

Die Abbildung [4.11] illustriert die eingesetzte Technik bei der Erzeugung der Rundenschlüssel. Diese Rundenschlüssel werden in einem t-word Array abgelegt:

```
S[0], S[1], ... S[t-1]
```

Die beiden Parameter w und r sind Inputparameter des Algorithmus zur Erzeugung der Subkeys. Der Array `S[]`
is initialized to a particular fixed pseudorandom bit pattern. The the b-byte key `K[0, ... b-1]` is converted into a c word array

```
L[0], L[1], ... L[c-1]
```

On a little endian machine this is achieved by zeroing out the array `L[]` and copying the string `K` directly into the memory positions represented by `L[]`. If b is not an integer multiple of w, then a portion of `L[]` at the right'end remains zero. Finally a mixing operation is performed that applies the content of `L[]` to the initialized value of `S[]` to produce a final value for the array `S[]`.

Im Detail funktioniert das Verfahren wie folgt:

Die Initialisierung verwendet zwei Konstanten der Länge eines Words, die fol-

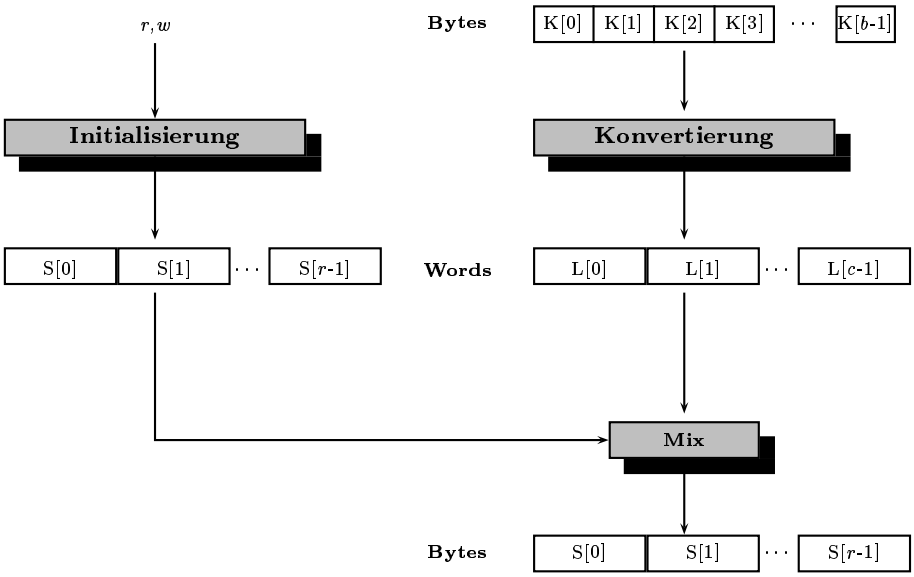


Abbildung 4.11: Ableitung der Rundenschlüssel im RC5 Algorithmus.

gendermaßen definiert sind:

$$P_w = \texttt{Odd}[(e-2)\cdot 2^w]$$
$$Q_w = \texttt{Odd}[(\phi-1)\cdot 2^w]$$

wobei

$$e = 2.718281828459\ldots \qquad \text{EULER Konstante}$$
$$\phi = 1.618033988749\ldots \qquad \text{Goldener Schnitt}$$

und $\texttt{Odd}[x]$ bezeichnet die ungerade ganze Zahl, die am nächsten (kleiner oder größer bei x liegt. Beispielsweise

$$\text{Odd}[e] = 3$$
$$\text{Odd}[\phi] = 1$$

Nun sind die erlaubten Werte der Parameter w: 16, 32 bzw. 64. Daher ergeben sich folgende Konstanten:

w	**16**	**32**	**64**
P_w	0xB7E1 = 47073	B7E15163	B7E151628AED2A6B
Q_w	0x9E37 = 40503	9E3779B9	9E3779B97F4A7C15

Check:

$$\begin{aligned}
P_w^{16} &= \texttt{Odd}\Big[(e-2)\cdot 2^{16}\Big] \\
&= \texttt{Odd}\Big[(e-2)\cdot 65536\Big] \\
&= \texttt{Odd}\Big[0.718281828459\cdot 65536\Big] \\
&= \texttt{Odd}\Big[0.718281828459\cdot 65536\Big] \\
&= \texttt{Odd}\big(47073.3179099\big) \\
&= 47\,073 \\
&= 0xB7E1
\end{aligned}$$

Analog:

$$\begin{aligned}
Q_w^{16} &= \texttt{Odd}\Big[(\phi-1)\cdot 2^{16}\Big] \\
&= \texttt{Odd}\Big[(\phi-1)\cdot 65536\Big] \\
&= \texttt{Odd}\Big[0.618033988749\cdot 65536\Big] \\
&= \texttt{Odd}\Big[40503,475\ldots\Big] \\
&= 40\,503 \\
&= 0x9E37
\end{aligned}$$

Die anderen Werte obiger Tabelle leiten sich analog ab.

Mit diesen beiden Konstanten P_w und Q_w wird der Array `S[]` folgendermaßen initialisiert:

$$\begin{aligned}
&\texttt{S[0]} = P_w; \\
&\text{for } i = 1 \text{ to } t-1 \text{ do} \\
&\qquad \texttt{S[i]} = \texttt{S[i}-1\texttt{]} + Q_w;
\end{aligned}$$

Hier wird die Addition modulo 2^w ausgeführt. The initialized array `S[]` is then mixed with the key array `L[]` to produce a final array `S[]` of subkeys. To achieve this goal, three passes are made through the larger of the two arrays; the smaller array may be handled more times:

$$\begin{aligned}
&i = j = X = Y = 0 \\
&\text{do } 3 \times \texttt{max}(t,c) \text{ times} \\
&\qquad \texttt{S[i]} = (\texttt{S[i]} + X + Y) \lll 3; \\
&\qquad i = (i+1) \bmod t; \\
&\qquad \texttt{L[j]} = (\texttt{L[j]} + X + Y) \lll (X+Y); \\
&\qquad Y = \texttt{L[j]}; \\
&\qquad j = (j+1) \bmod c;
\end{aligned}$$

here the operation $\lll$ means left circulation, see below.

4.4.3 The RC5 Encryption Algorithm

The RC5 encryption scheme uses three primitive operations and their inverses:

- **Addition**
  Addition of words – denoted by the + sign – is performed modulo 2^w. The inverse operation – denoted by the - sign – is subtraction modulo 2^w.
- **XOR**
  The bitwise exclusive OR (=XOR) is denoted by the $\oplus$ sign.
- **Left shift**
  The cyclic rotation of word x left by y bits is denoted by $x <<< y$. The inverse is the right circular shift, denoted by $x >>> y$.

Figure 4.12 depicts the encryption operation. Note that this is not a classical FEISTEL structure. The plaintext is assumed to initially reside in the two registers A and B, both have a size of w bits. We denote by LE_i and RE_i the left and right half of the data when round i has completed. Thus we can formulate the RC5 encryption algorithm as follows:

$$
\begin{aligned}
&\mathrm{LE}_0 = A + \mathtt{S}[0] \\
&\mathrm{RE}_0 = B + \mathtt{S}[1] \\
&\text{for } i = 1 \text{ to } r \text{ do} \\
&\qquad \mathrm{LE}_i = ((\mathrm{LE}_{i-1} \oplus \mathrm{RE}_{i-1}) <<< \ \mathrm{RE}_{i-1}) + \mathtt{S}[2 \cdot \mathtt{i}] \\
&\qquad \mathrm{RE}_i = ((\mathrm{RE}_{i-1} \oplus \mathrm{LE}_{i-1}) <<< \ \mathrm{LE}_{i-1}) + \mathtt{S}[2 \cdot \mathtt{i} + 1]
\end{aligned}
$$

The resulting ciphertext is contained in the two variables LE_r and RE_r. Each of the r rounds consists of

- a substitution ($\oplus$ operation) using both words of the data
- a permutation operation (the left shift) using both words of data
- a further substitution (+ operation) depending on the key

Note the simplicity of this algorithms; there are only five lines of program code. Also note that both halves of the data are updated in each round. Thus one round in RC5 has a similar effect as two DES-rounds.

4.4.4 RC5 Entschlüsselung

The RC5 Decryption algorithm is shown in Figure 4.13. Here the $2w$ bits of ciphertext are initially assigned to the two one-word variables LD_r and RD_r. Here, LD_i and RD_i refer to the left and right part of the data before the ith round. The rounds are numbered from r down to 1.

$$
\begin{aligned}
&\text{for } i = r \text{ to } 1 \text{ do} \\
&\qquad \mathrm{RD}_{i-1} = ((\mathrm{RD}_i - \mathtt{S}[2 \cdot \mathtt{i} + 1] >>> \ \mathrm{LD}_i) \oplus \ \mathrm{LD}_i) \\
&\qquad \mathrm{LD}_{i-1} = ((\mathrm{LD}_i - \mathtt{S}[2 \cdot \mathtt{i}] >>> \ \mathrm{RD}_{i-1}) \oplus \ \mathrm{RD}_{i-1}) \\
&A = \mathrm{RD}_0 - \mathtt{S}[1] \\
&B = \mathrm{LD}_0 - \mathtt{S}[0]
\end{aligned}
$$

The two most striking features of RC5 are

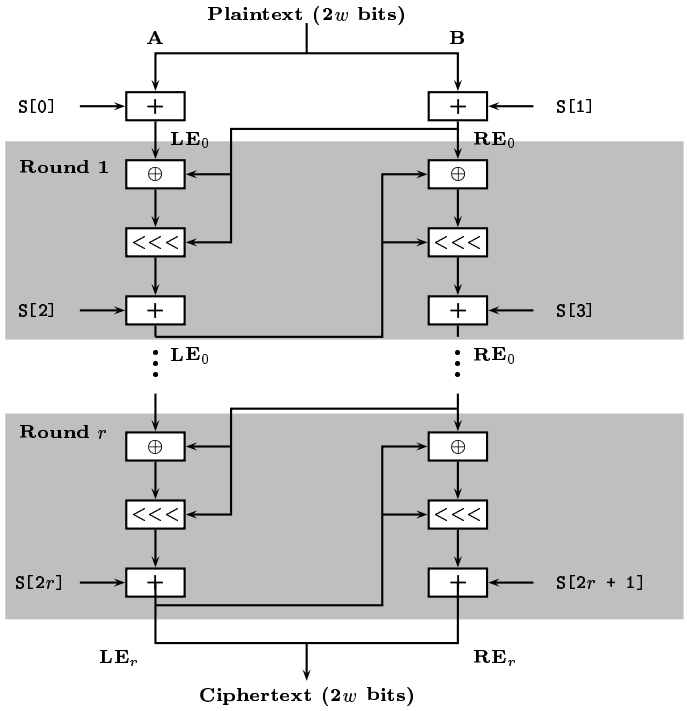


Abbildung 4.12: RC5 Encryption Algorithm.

- the simplicity of the algorithm
- the use of data-dependent rotations.

The rotations are the only nonlinear portion of the algorithm.

4.4.5 Modes

To improve the effectiveness of RC5 in interoperable implementations, RC5 comes with four different modes of operation:

- **RC5 block cipher**
  This is the raw encryption algorithm that takes a fixed-size input block – $2w$ bits – and produces a ciphertext block of the same length using a key-dependent transformation. This is also known as the *electronic codebook mode* (ECB).

- **RC5-CBC**
  This is the cipher block chaining mode (CBC) for RC5. CBC processes messages whose length is a multiple of the RC5 block size (i.e. multiples of $2w$ bits). CBC provides improved security to ECB because repeated blocks of plaintext produce different blocks of ciphertext.

- **RC5-CBC-Pad**
  This is a CBC style of algorithm that handles plaintext of any length. The ciphertext will be longer than the plaintext by at most the size of a single RC5 block.

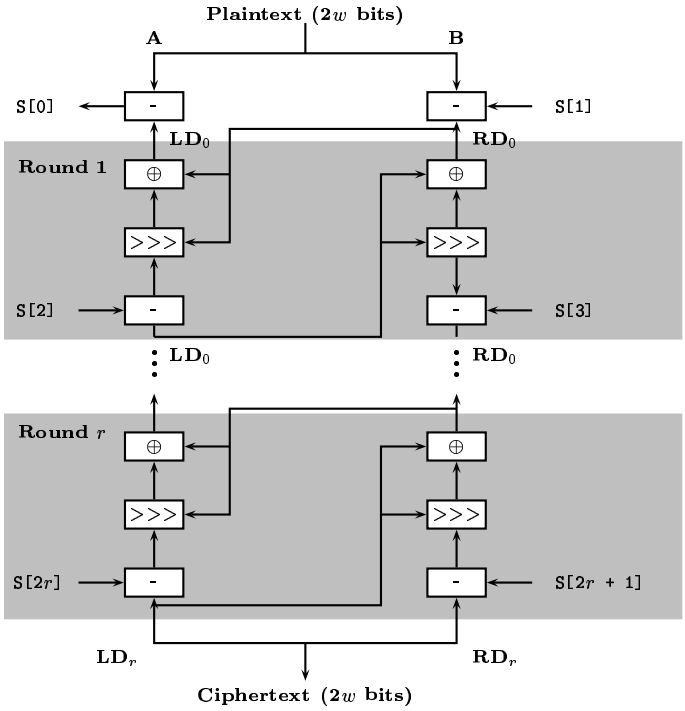


Abbildung 4.13: RC5 Decryption Algorithm.

- **RC5-CTS**
  This denotes the ciphertext stealing mode which is also a CBC style. This mode handles plaintext of any length and produces ciphertext of equal length.

Let us give a few remarks on the two last modes.

When a CBC mode is used to encrypt messages, some technique is always needed to cope with messages that are not a multiple of the block length. The simples approach is to use padding. In RC5, it is assumed that the message is an integer number of bytes. At the end of the message, from 1 to bb bytes of padding are added, where bb equals the block size for RC5 measured in bytes (i.e. $bb = 2w/8$). The pad bytes are all the same and are set to a byte that represents the number of bytes of padding. For example, if there are 8 bytes of padding, each byte has the bit pattern 0000 1000.

Padding may not always be appropriate. For example, one might wish to store the encrypted data in the same memory buffer that originally contained the plaintext. In that case, the ciphertext must have the same length as the plaintext. This is achieved by the CTS mode.

Kapitel 5

Der Advanced Encryption Standard – AES

Im Jahre 1997 initialisierte das NIST die Suche nach einem neuen Verschlüsselungsstandard als Nachfolger des Data Encryption Standards DES. Dieser neue Verschlüsselungsstandard heisst **Advanced Encryption Standard** oder kurz **AES**. Der neue Verschlüsselungsstandard wurde vom NIST öffentlich ausgeschrieben, in einem iterativen Verfahren wurden verschiedene Kandidaten eingereicht, ausgewählt und in allen Details diskutiert. Das NIST formulierte folgende Anforderungen an den zukünftigen Standard:

- **Formale Aspekte**
  Der neue Standard muß eine symmetrische Blockchiffre sein mit einer Blocklänge von 128 Bits und variabler Schlüssellänge von 128, 192 und 256 Bits.

- **Patentrechtliche Aspekte**
  Der Algorithmus, der als zukünftiger AES zertifiziert wird, muss patentfrei sein.

- **Sicherheit**
  Der AES muss Sicherheit gewährleisten gegenüber allen bekannten Angriffsarten. Diese Sicherheit muss mathematisch beweisbar sein.

- **Einfachheit**
  Das Design des Algorithmus muss einfach und nachvollziehbar sein.

- **Flexibilität**
  Der AES muss flexibel einsetzbar sein, sowohl in Hinblick auf die Blocklänge als auch auf die Wahl der Schlüssellänge.

- **Effizienz**
  Der AES muss effizienter sein als Triple–DES.

- **Implementierung**
  Die Hardware und Software Implementierung des neuen Algorithmus muss einfach sein.

Seit dem 2. Oktober 2000 ist der Gewinner des Auswahlverfahrens bekannt, der **Rijndael** Algorithmus, der von **Joan Daemen** und **Vincent Rijmen** entwickelt wurde. Das Design des Rijndael Algorithmus ist ausführlich in der Monographie [54] beschrieben.

Abbildung 5.1: Joan Daemon (links) und Vincent Rijmen.

Hier einige Highlights der Entwicklung des AES in Stichpunkten:

2.1.1997 NIST kündigt Vorschläge für den AES an.

12.9.1997 Aufruf vom NIST an die Kryptogemeinde, adäquate Algorithmen einzureichen.

20.8.1998 Erste AES Konferenz zur Vorstellung von 15 AES Kandidaten, das NIST bittet um Kommentare.

15.4.1999 Die Kandidaten für die Endauswahl werden vorgestellt. Diese sind

- MARS von IBM (siehe [260])
- RC6 von Ron Rivest
- Rijndael von Joan Daemen und Vincent Rijmen
- Serpent von Ross Anderson, Eli Biham und Lars Knudsen
- Twofish von Bruce Schneier und anderen.

13.-14.4.2000 Dritte AES Konferenz (AES2) mit öffentlicher Diskussion der fünf Algorithmen.

15.5.2000 Deadline für öffentliche Kommentare.

2.10.2000 The winner is.... Rijndael.

April-June 2001 Der AES wird offiziell FIPS *federal information processing standard*

26.11.2001 NIST publiziert den FIPS-197 mit den Details des Advanced Encryption Standards.

Am **19. Mai 2005** kündigt das NIST das offizielle Ende des DES an und zieht die Empfehlung FIPS-PUB 46 zurück.

Im Jahre 2002 wird eine vielversprechende Methode präsentiert, um den Advanced Encryption Standard zu kompromittieren (siehe [234]).

5.1 Vereinfachter Rijndael–Algorithmus

Ein vereinfachter Rijndael–Algorithmus — wir nennen diesen Algorithmus S–AES für Simplified AES — wurde von EDWARD SCHAEFER und seinen Studenten entwickelt [159].[1] Der S–AES Algorithmus ist kein sicheres Verschlüsselungsverfahren, er dient dazu, anhand einfacher Parameter die Grundstruktur des Rijndael–Algorithmus zu demonstrieren.

5.1.1 Überblick

Die Abbildung [5.2] zeigt die komplette Struktur des S–AES–Algorithmus.

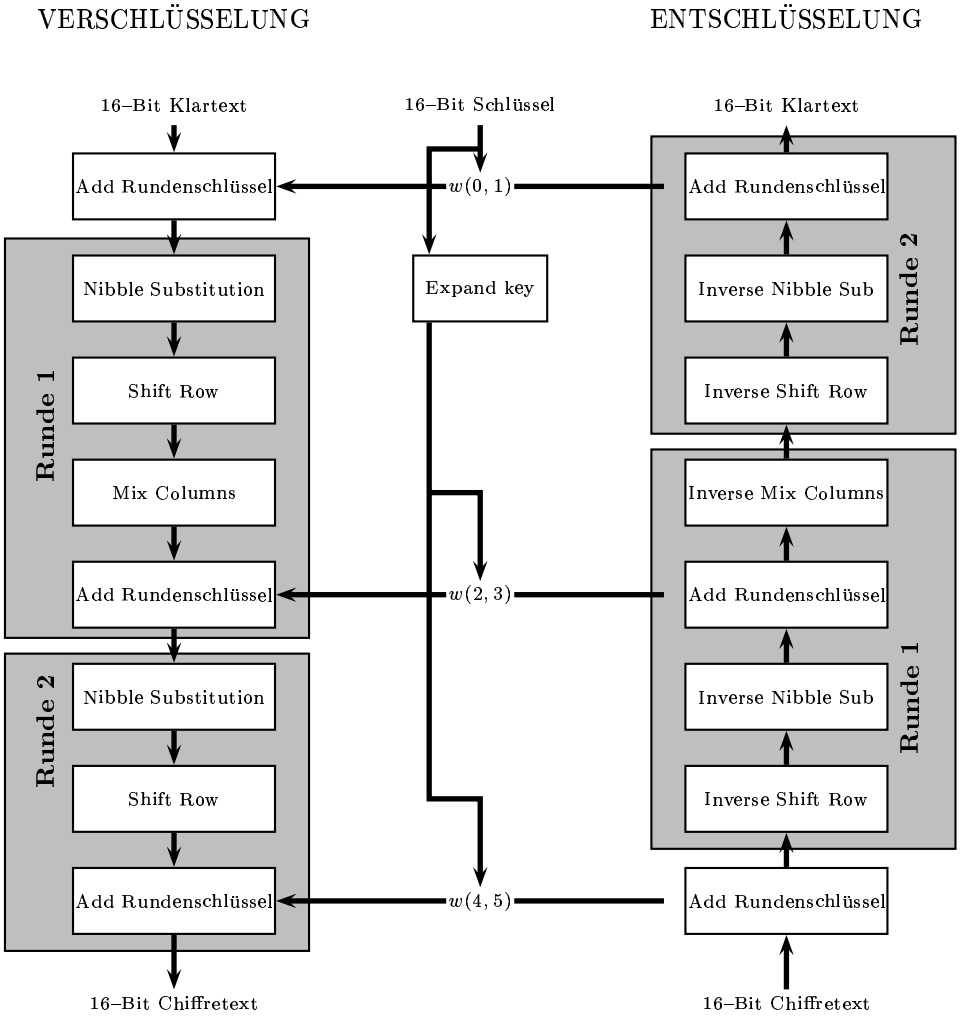


Abbildung 5.2: Ablaufschema der Ver– bzw. Entschlüsselung des S–AES Algorithmus.

Der Verschlüssungsalgorithmus hat als Input einen 16–Bit Klartextblock und

[1] Siehe auch Appendix 5B in STALLINGS [207] oder [168].

einen 16–Bit Schlüssel und produziert einen 16–Bit Chiffreblock. Der S–AES Entschlüsselungsalgorithmus hat einen 16–Bit Chiffreblock und den gleichen 16–Bit Schlüssel als Input und liefert den ursprünglichen 16–Bit Klartextblock als Output.

Die Bausteine des Verschlüsselungsalgorithmus sind vier unterschiedliche Funktionen oder Transformationen

- Add Rundenschlüssel (A_K),
- Nibbel Substitution (NS),
- Shift Row (SR),
- Mix Column (MC),

deren Funktionalitäten später noch näher untersucht werden.

Wir können den Verschlüsselungsalgorithmus kompakt als Komposition von Funktionen schreiben:

$$E_{S--AES} = A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0},$$

wobei die Operation A_{k_0} zuerst angwendet wird.

der Verschlüsselungsalgorithmus ist in drei Runden unterteilt, Runde 0 ist einfach eine Add Key Operation. Die Runde 1 ist eine Runde, die aus der Anwendung von vier Funktionen besteht und Runde 2 besteht aus der Anwendung von drei operationen. Jede Runde beinhaltet die Add Rundenschlüssel Operation, die einen 16–Bit Rundenschlüssel verwendet. Der ursprüngliche 16–Bit Schlüssel wird erweitert auf 48 Bit, so dass in jeder Runde ein unterschiedlicher Rundenschlüssel K_0, K_1, K_2 verwendet wird.

Jede Funktion operiert auf einem 16–Bit **Zustand**, der als 2×2 Matrix von Nibbles betrachtet wird; wobei ein Nibble aus 4 Bit besteht. Der initiale Wert der Zustandsmatrix ist der 16–Bit Klartext. Jede Funktion des Verschlüsselungsalgorithmus modifiziert diesen Zustand und liefert nach Anwendung der letzten Funktion den 16–Bit Ciffretext.

Abbildung 5.3: Zustandsmatrix in verschiedenen Darstellungen.

Wie in der Abbildung [5.3] dargestellt ist, werden die Nibbles in der Matrix in Form von Spalten geordnet. Die ersten 8 Bit eines 16–Bit Klartextblocks,

der als Input in den Verschlüsselungsalgorithmus dient, belegen daher die erste Spalte der Matrix, die zweiten 8 Bits belegen die zweite Spalte. Der 16–Bit Schlüssel wird in ähnlicher Weise organisiert, es ist jedoch vorteilhafter diesen Schlüssel als 2 Bytes zu betrachten als 4 Nibbles, siehe die Abbildung [5.4]. Der erweiterte Schlüssel, der aus 48 Bit besteht, liefert die drei Rundenschlüssel, die Bits werden wie folgt bezeichnet:

$$K_0 = k_0 k_1 \dots k_{15},$$
$$K_1 = k_{16} k_{17} \dots k_{31},$$
$$K_2 = k_{32} k_{17} \dots k_{47}.$$

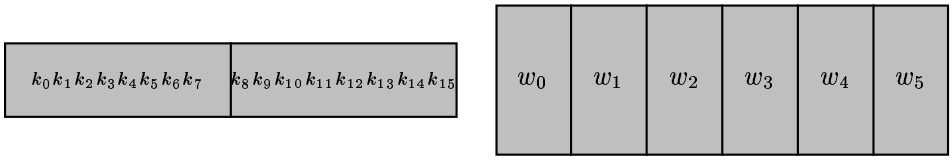


Abbildung 5.4: S–AES Datenstrukturen für den Schlüssel.

Die Abbildung [5.5] zeigt die wesentlichen Elemente einer kompletten Runde des S–AES Algorithmus.

Die Abbildung [5.2] enthält auch die Schritte des Entschlüsselungsverfahrens und ist im wesentlichen die Umkehrung der Verschlüsselung:

$$D_{S--AES} = A_{K_0} \circ INS \circ ISR \circ IMC \circ A_{K_1} \circ INS \circ ISR \circ A_{K_2},$$

wobei drei der Funktionen entsprechende Umkehrfunktionen besitzen: Inverse Mix Column (IMC), Inverse Nibble Substitution (INS) und Inverse Shift Row (ISR).

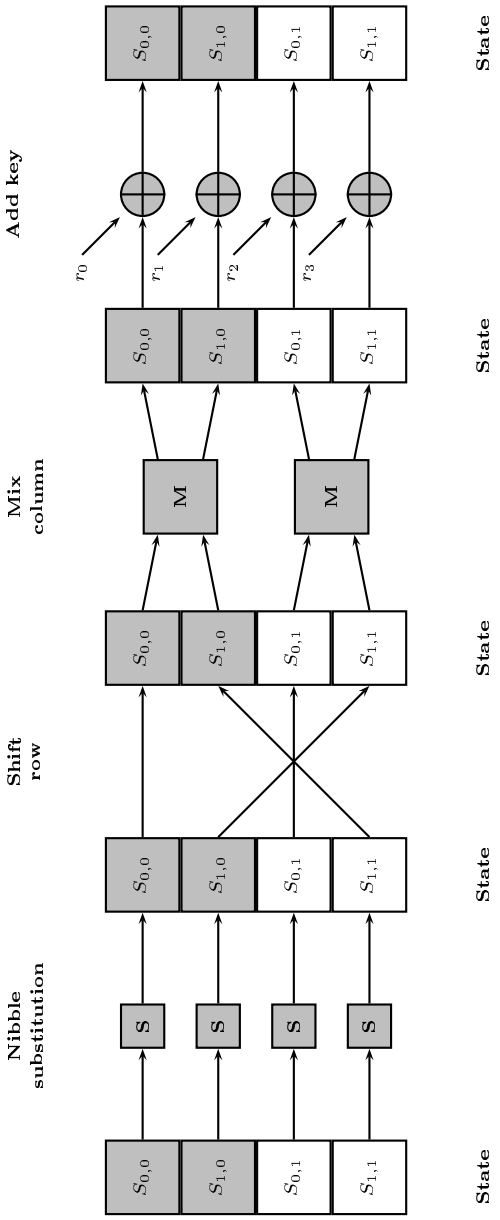


Abbildung 5.5: Eine S–AES Verschlüsselungsrunde.

5.1.2 S–AES Ver– und Entschlüsselung

Wir sehen uns nun die einzelnen Funktionen im Detail an, die bei der Verschlüsselung angewendet werden.

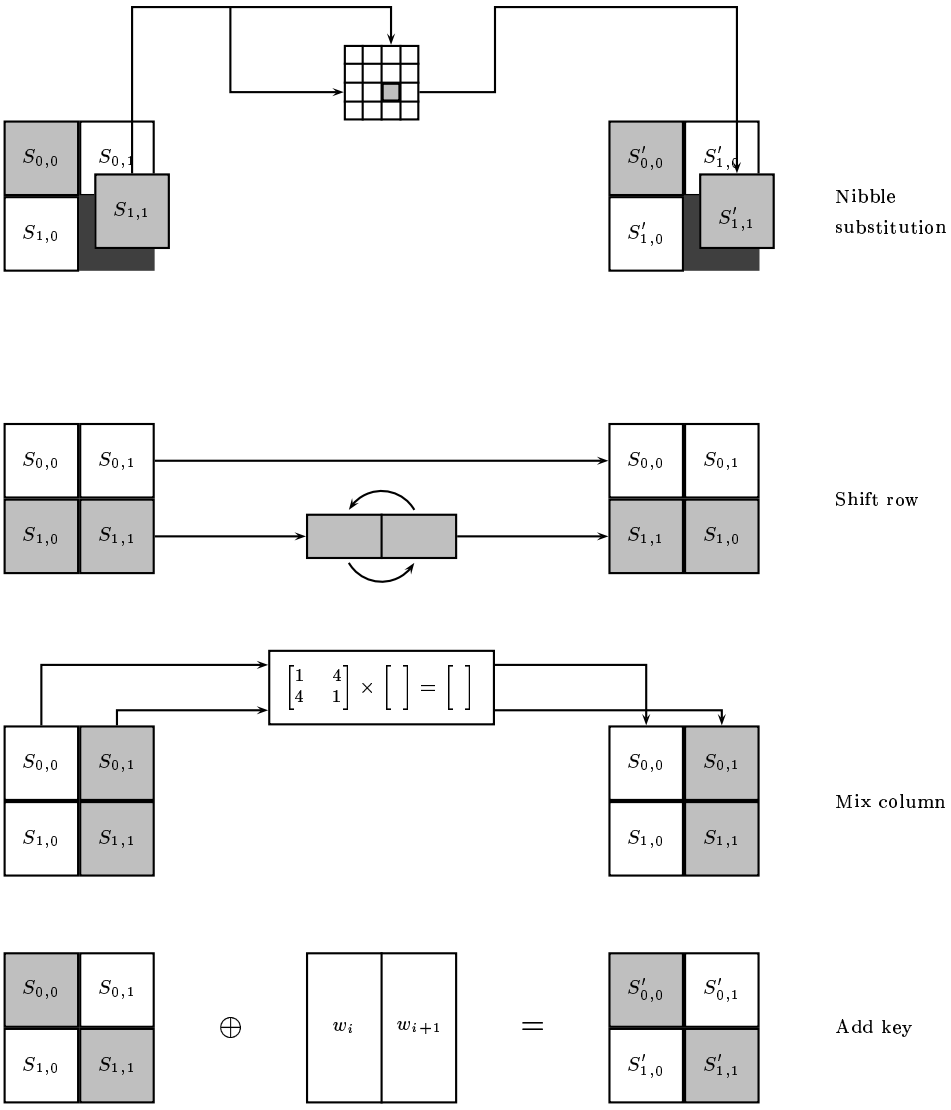


Abbildung 5.6: Operation bei der S–AES Verschlüsselung.

1. **Die Add–Key Operation**
   Die Add–Key Operation besteht aus der bitweisen XOR–Verknüpfung der 16–Bit Zustandsmatrix und dem 16 Bit Schlüssel. Die folgende Abbildung [5.7] zeigt ein Beispiel dieser Operation.

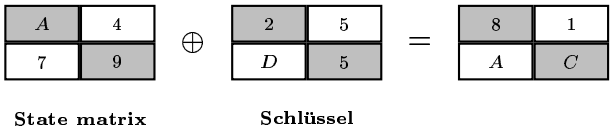


Abbildung 5.7: Details der Add–Key Operation.

In der Abbildung [5.6] ist dieser Schritt als spaltenorientierte Operation dargestellt, sie kann jedoch auch als nibbleweise oder bitweise Transformation aufgefasst werden.

Die Inverse der Add–Key Operation ist identisch zu der Add–Key Operation, da für die XOR–Operation gilt:

$$a \oplus a \oplus b = b.$$

2. **Die Nibble–Substitutionsoperation**
Die Nibble–Substitutionsoperation ist eine einfache Table–Lookup Funktion, siehe Abbildung [5.6]. Im AES wird eine 4×4–Matrix definiert, deren Werte Nibbles sind. Diese Matrix nennt man **S–Box**, sie enthält eine Permutation aller möglichen 4–Bit Werte. Jedes individuelle Nibble eines Zustands wird in folgender Weise auf ein neues Nibble abgebildet. Die beiden links stehenden Bits des Nibbles werden als Zeilennummer gelesen, die beiden rechts stehenden Bits als Spaltennummer. Dies Zeilen– und Spaltenwerte indizieren ein Element der S–Box und wählen auf diese Weise einen eindeutigen 4–Bit Output fest.

		j			
		00	01	10	11
i	00	**9**	**4**	**A**	**B**
	01	**D**	**1**	**8**	**5**
	10	**6**	**2**	**0**	**3**
	11	**C**	**E**	**F**	**7**

(a) S–Box

		j			
		00	01	10	11
i	00	**A**	**5**	**9**	**B**
	01	**1**	**7**	**8**	**F**
	10	**6**	**0**	**2**	**3**
	11	**C**	**4**	**D**	**E**

(b) Inverse S–Box

Abbildung 5.8: Die S–AES S–Boxen. Die grau unterlegten Zellen enthalten Hexadezimalzahlen, die weißen Zellen enthalten Binärzahlen.

Beispielsweise wird der Hexadezimalwert **A** durch die S–Box auf den Wert 0 abgebildet, denn es ist

$$\mathbf{A} = 1010_2,$$

daher referenziert dies die Zeile 2 und Spalte 2 der S–Box. Das S_{22} Element der Matrix — siehe dazu die S–Box in der Abbildung [5.8] — ist die 0, daher ist der Output dieser Operation 0000.

Beispiel [5.12]

Betrachten wir als Input den Zustand

A	*C*
8	1

Dann werden die Nibbles dieses Zustands durch die S–Bos wie folgt abgebildet:

8 = 10 00	2. Zeile, 0. Spalte	$\longrightarrow$	6	=	01 10
A = 10 10	2. Zeile, 2. Spalte	$\longrightarrow$	0	=	00 00
C = 11 00	3. Zeile, 0. Spalte	$\longrightarrow$	C	=	11 00
1 = 00 01	0. Zeile, 1. Spalte	$\longrightarrow$	4	=	01 00

Die Substitutionsoperation lässt sich daher kompakt darstellen in der Form

A	*C*
8	1

$\longrightarrow$

0	*C*
6	4

Die inverse Nibble Substitution verwendet die inverse S–Box, die in der Abbildung [5.8] abgebildet ist. Wie man ablesen kann, produziert der Input 0 den Output *A*, der Input *A* der S–Box (links) liefert den Output 0.

3. **Die Shift–Row Operation**
   Die Shift–Row Operation führt einen ein–Nibbel Shift durch in der zweiten Zeile der Zusandsmatrix, die erste Zeile wird nicht modifiziert.

B	2
3	*A*

$\longrightarrow$

B	2
A	3

Die inverse Shift Row Funktion ist mit der Shift Row Operation identisch.

Die **Mix–Column Operation**

Die Mix–Column Funktion ist die komplizierteste Operation im S–AES Algorithmus und erfordert eine etwas nähere Untersuchung. Diese Funktion operiert auf jede Spalte des Zustands einzeln. Jeder Nibble einer Spalte wird auf einen neuen Wert abgebildet, der eine Funktion beider Nibbles der Inputspalte ist. Die Transformation kann als Matrizenprodukt auf Zustände definiert werden (siehe auch die Abbildung [5.6]):

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}.$$

Führt man die Matrizenmultiplikation aus, ergibt sich:

$$\begin{aligned} S'_{0,0} &= S_{0,0} \oplus (4 \cdot S_{1,0}), \\ S'_{1,0} &= (4 \cdot S_{0,0}) \oplus S_{1,0}, \\ S'_{0,1} &= S_{0,1} \oplus (4 \cdot S_{1,1}), \\ S'_{1,1} &= (4 \cdot S_{0,1}) \oplus S_{1,1}. \end{aligned}$$

Die Arithmetik wird auf dem endlichen Körper $GF\left(2^4\right)$ ausgeführt, die Verknüpfung $\cdot$ steht für die Multiplikation in dem Körper $GF\left(2^4\right)$.

Arithmetik in $GF\left(2^4\right)$

Die Arithmetik auf der Körpererweiterung $GF\left(2^4\right)$ ist ausführlich in Abschnitt [22.4] untersucht, wir fassen sie hier zusammen. Die Tabelle [5.1] zeigt die Additionsoperation, im wesentlichen ist dies die XOR Verknüpfung, *e.g.*

$$\begin{aligned} 6 \oplus A &= \texttt{0110} \oplus \texttt{1010} = \texttt{1100} = C, \\ 3 \oplus 7 &= \texttt{0011} \oplus \texttt{0111} = \texttt{0100} = 4. \end{aligned}$$

Die Multiplikation ist in $GF\left(2^4\right)$ mod x^4+x+1. Wir sehen uns diese Operation an dem folgenden Beispiel explizit an.

$$3 \cdot A = \texttt{0011} \cdot \texttt{1010}.$$

In der Terminologie der Polynome ist dies das Produkt

$$(x+1) \cdot (x^3+x) \bmod (x^4+x+1).$$

Nun ist (gewöhnliche Multiplikation)

$$(x+1) \cdot (x^3+x) \bmod (x^4+x+1) = (x^4+x^3+x^2+x) \bmod (x^4+x+1).$$

Weil der Grad des Produktpolynoms größer oder gleich gem Grad des Modulusploynoms, muss eine Polynomdivision durchgeführt werden, um die Reste zu finden.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Tabelle 5.1: Addition auf $GF(2^4)$ mod 2.

Es ist[2]

$$\begin{array}{rrrrrrrrrrrrrrrr} x^4 & + & x^3 & + & x^2 & + & x & & : & x^4 & + & x & + & 1 & = 1 \\ x^4 & & & & & + & x & +1 & : & & & & & \\ \hline & & x^3 & + & x^2 & & & +1 & & & & & & \end{array}$$

Somit verbleibt als Rest das Polynom x^3+x^2+1. Im Binärsystem korrespondiert dies mit `1101`, oder Hexadezimal D. Daher ist

$$3 \cdot A = D.$$

Die vollständige Multiplikationstabelle auf $GF\left(2^4\right)$ mod x^4+x+1 ist in der Tabelle [5.2] angegeben

---

Die Mix–Column Funktion operiert auf jeder einzelnen Spalte des Zustands. Jedes Nibble einer Sspalte wird auf einen neuen Wert abgebildet, der wiederum eine Funktion der beiden Nibble der Spalte ist. Wir haben die Transformation wie folgt definiert

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix}.$$

Wir schreiben dies um in die Terminologie mit Polynomen. Der Wert 1 entspricht dem Polynom 1, der (binäre) Wert $4 =$ `0100` korrespondiert mit x^2. Daher

$$\begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix}.$$

[2]Die Addition ist wieder mod 2.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A

Tabelle 5.2: Multiplikation auf $GF(2^4)$ mod $x^4 + x + 1$.

die Multiplikation wird ausgeführt modulo $x^4 + x^2 + 1$. In der Terminologie der Polynome kann eine einfache Erklärung für die Arithmetik angegeben werden. In der Bit–Notation (siehe Abbildung [5.3]) können die Mix–Column Opeartion wie folgt darstellen:

$$\begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix} \begin{bmatrix} b_0x^3 + b_1x^2 + b_2x + b_3 & b_8x^3 + b_9x^2 + b_{10}x + b_{11} \\ b_4x^3 + b_5x^2 + b_6x + b_7 & b_{12}x^3 + b_{13}x^2 + b_{14}x + b_{15} \end{bmatrix}.$$

5.2 Das Rijndael Kryptosystem

Rijndael ist eine iterative Block Chiffre mit variabler Blocklänge. Die Länge der Blöcke — diese bezeichnen wir mit b — und die Schlüssellänge k können unabhängig voneinander auf einen der drei Werte 128, 192 oder 256 Bits gesetzt werden. Die Anzahl der Runden — diesen Parameter bezeichnen wir mit r — variiert zwischen 10 und 14, abhängig von der Block– bzw. Schlüssellänge. Der Zusammenhang zwischen diesen Parametern des Rijndael Kryptosystems ist in der Tabelle [5.3] dargestellt.

r	$b = 128$	$b = 192$	$b = 256$
$k = 128$	10	12	14
$k = 192$	12	12	14
$k = 256$	14	14	14

Tabelle 5.3: Anzahl der Runden im Rijndael Algorithmus in Abhängigkeit von der Blockgröße b und Schlüssellänge k.

Wir diskutieren hier den Fall: Blocklänge 128 Bits und Schlüssellänge 192 Bits.

Die Zwischenschritte des Rijndael Verschlüssellungsverfahrens werden *States* (Zustände) genannt und mit S_i bezeichnet. Ein State hat eine Länge von 128 Bits, die in 16 Bytes gruppiert werden. Diese 16 Bytes bezeichnen wir mit $a_{k,l}^{(i)}$ und gruppieren sie in Matrixform:

$$S_i = \begin{pmatrix} a_{0,0}^{(i)} & a_{0,1}^{(i)} & a_{0,2}^{(i)} & a_{0,3}^{(i)} \\ a_{1,0}^{(i)} & a_{1,1}^{(i)} & a_{1,2}^{(i)} & a_{1,3}^{(i)} \\ a_{2,0}^{(i)} & a_{2,1}^{(i)} & a_{2,2}^{(i)} & a_{2,3}^{(i)} \\ a_{3,0}^{(i)} & a_{3,1}^{(i)} & a_{3,2}^{(i)} & a_{3,3}^{(i)} \end{pmatrix}.$$

Jede Spalte dieser Matrix hat eine Länge von 32 Bits, was der Länge eines *Words* entspricht.

Daher kann ein State als rechteckiger Array von Bytes dargestellt werden. Im allgemeinen hat dieser Array vier Zeilen, die Anzahl der Spalten — die mit Nb bezeichnet wird — ist gleich der Blocklänge dividiert durch die Wortlänge (32 Bit). Die folgende Tabelle zeigt diesen Zusammenhang:

Block length	`Nb`
128	4
192	6
256	8

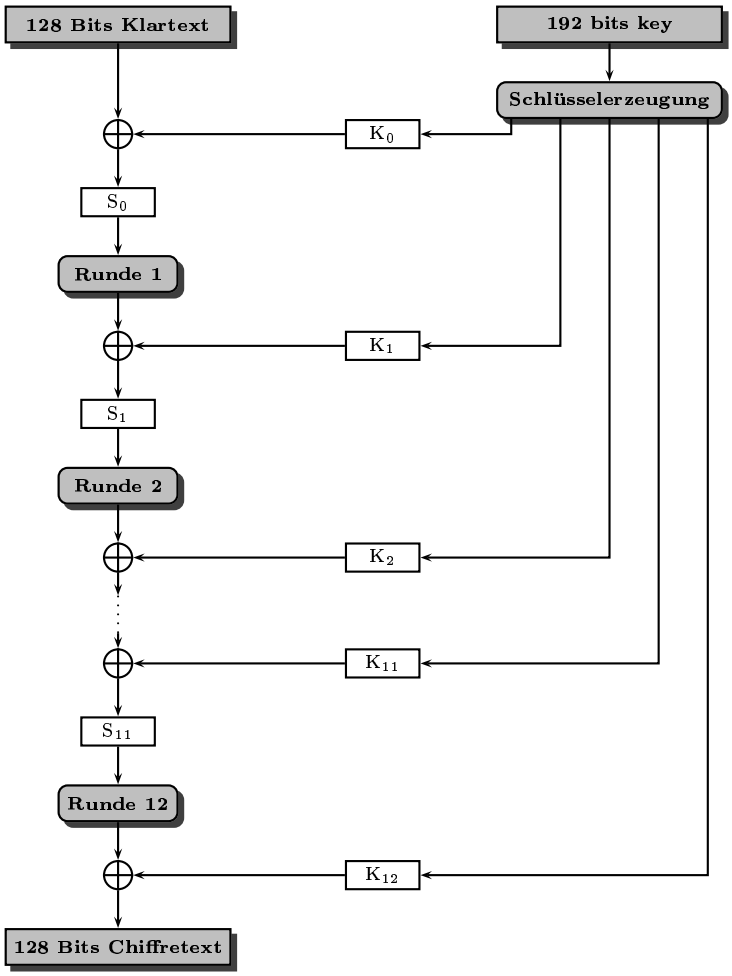


Abbildung 5.9: Verschlüsselung mit dem AES Kryptosystem. In diesem Beispiel ist die Blocklänge 128 Bits und die Schlüssellänge ist 192 Bits.

Den Schlüssel kann man auf ähnliche Weise in Form eines rechteckigen Arrays mit vier Zeilen darstellen. Im allgemeinen ist die Anzahl der Spalten des Schlüssels — diese wird mit Nk bezeichnet — gleich der Schlüssellänge dividiert durch die Wortlänge 32 Bits.

Daher habe wir für unser Beispiel mit einer Schlüssellänge von 192 Bits eine Gruppierung des Keys in sechs Spalten:

$$K_i = \begin{pmatrix} k_{0,0}^{(i)} & k_{0,1}^{(i)} & k_{0,2}^{(i)} & k_{0,3}^{(i)} & k_{0,4}^{(i)} & k_{0,5}^{(i)} \\ k_{1,0}^{(i)} & k_{1,1}^{(i)} & k_{1,2}^{(i)} & k_{1,3}^{(i)} & k_{1,4}^{(i)} & k_{1,5}^{(i)} \\ k_{2,0}^{(i)} & k_{2,1}^{(i)} & k_{2,2}^{(i)} & k_{2,3}^{(i)} & k_{2,4}^{(i)} & k_{2,5}^{(i)} \\ k_{3,0}^{(i)} & k_{3,1}^{(i)} & k_{3,2}^{(i)} & k_{3,3}^{(i)} & k_{3,4}^{(i)} & k_{3,5}^{(i)} \end{pmatrix}.$$

Daher ist jede Spalte ein Array der Länge von 32 Bits, also ewin 'Word'.

Bevor die erste Verarbeitungsrunde durchlaufen wird, wird der Klartext und der Schlüssel in 32 Bit Blöcke in der Zustandsmatrix bzw. Schlüsselmatrix abgelegt.

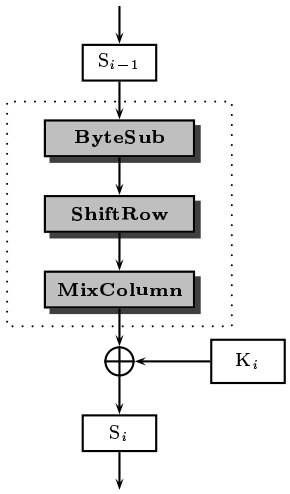


Abbildung 5.10: Details einer einzelnen Runde.

The input and output of Rijndael at its external interface are considered to be one dimensional arrays of 8-bit bytes numbered upwards from 0 to 4·`Nb` - 1. Thus, depending on the choice of block length, these arrays have lengths 16, 24 or 32 bytes and array indices in the ranges: $0 \ldots 15; 0, \ldots 23$ or $0 \ldots 31$. The cipher key is considered to be a one-dimensional array of 8-bit bytes numbered upwards from 0 to 4·`Nk` - 1. These blocks hence have lengths of 16, 24 or 32 bytes and array indices in the ranges $0 \ldots 15; 0 \ldots 23$ or $0 \ldots 31$.

The cipher input bytes – i.e. the plaintext – are mapped onto the state bytes in the order

$$a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, a_{1,1}, a_{2,1}, \ldots$$

and the bytes of the cipher key are mapped onto the array in the order

$$k_{0,0}, k_{1,0}, k_{2,0}, k_{3,0}, k_{0,1}, k_{1,1}, k_{2,1}, \ldots$$

At the end of the encryption operation, the cipher output is extracted from the state by taking state bytes in the same order.

In mathematical notation, if the one-dimensional index of a byte within a block is n and the matrix index is (i, j), then we have:

$$i = n \bmod 4; \quad j = \lfloor \frac{n}{4} \rfloor \quad n = i + 4 \cdot j;$$

The general Rijndael encryption scheme is depicted in Figure 5.9. In the configuration we are discussing, altogether there are 12 rounds. In each round, there is a subkey K_i resulting from the operation of *key expansion*. The key expansion generates $r+1$ subkeys $K_0, K_1 \ldots, K_r$ of length b from the k-bit key. The subkeys have the same length as the states. Before the first round and after each round the subkey is XORed with the actual state. The result of this operation is the input of the next round. Each round – except the last one – consists of the following four operations (see Figure 5.10):

- **ByteSub transformation**
- **ShiftRow transformation**
- **MixColumn transformation**
- **AddRoundKey transformation**

Note that the transformations of a single round do not depend on the key.

The specification of the component transformations are discussed in the following subsections.

5.2.1 Die ByteSub Transformation

The ByteSub transformation is the nonlinear S-box of AES. This transformation operates on each byte of the input state. The implementation of this transformation is a *table lookup* operation and corresponds to the calculation of the multiplicative inverse in the Galois field $GF(2^8)$. This operation is followed by the affine transformation

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

As usual all addition and multiplication operations are modulo 2. The inverse of the ByteSub transformation consists of the application of the inverse affine transformation followed by the calculation of the multiplicative onverse in $GF(2^8)$.

5.2.2 Die ShiftRow Transformation

Die `ShiftRow` Operation besteht aus einem zyklischen Shift der Zeilen 1 bis 3 der Zustandsmatrix. consists of a cyclic shift of the rows one to three of the state matrix. Row number 0 remains unchanged. Row 1 is shifted by c_1 bits to the right, row 2 shifted by c_2 bits and row number three is shifted cyclically by c_3 bits to the right. The values of c_1, c_2 and c_3 depend on the block length.

	$b = 128$	$b = 192$	$b = 256$
c_1	1	1	1
c_2	2	2	3
c_3	3	3	4

Tabelle 5.4: Values of the Right Shift in the ShiftRow Operation.

The inverse transormation – of course this is used for decryption – is a cyclic left shift.

5.2.3 Die MixColumn Transformation

In `MixColumn`, the columns of the state are interpreted as polynomials over the field $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, given by:

$$c(x) = \texttt{03}x^3 + \texttt{01}x^2 + \texttt{01}x + \texttt{02}$$

This polynomial is coprime to $x^4 + 1$ and therefore invertible. This can be represented as a matrix mutiplication. The `MixColumn` operation formally is a map that maps columns of the state into columns:

$$\begin{pmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{pmatrix} \longrightarrow \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{pmatrix}$$

The matrix elements are bytes in hexadecimal notation. Note that the matrix operation is on the field $GF(2^8)$.

5.2.4 The Round Key Addition

5.3 Key Schedule

As indicated in Figure 5.9 the key of the Rijndael chipher must be expanded in such a way that the input key of length 128, 192 or 256 bits maps to $r + 1$ round keys of b bits length. So, if a block length of 128 bits and 12 rounds are used, then altogether $128 \cdot 13 = 1.664$ bits for round keys are necessary.

The round keys are derived from the cipher key by means of the socalled *key schedule*. The key schedule has two components: the *key expansion* and the *round key selection*. The basic principle is the following:

- As noted above, the total number of round key bits is equal to the block length multiplied by the number of rounds + 1. (thus, for a block length of 128 bits and 10 rounds, 1.408 round key bits are nedded).
- The original cipher key is expanded into an *expanded key.*
- Round keys are taken from this expanded key in the following way: the first round key consists of the first `Nb` words, the second one of the following `Nb` words and so on.

Kapitel 6

Public Key Kryptosysteme

6.1 Einführung

Das Konzept der Public-Key Kryptographie entwickelte sich aus dem Versuch

Abbildung 6.1: Whitfield Diffie (links) und Martin E. Hellman.

heraus, zwei generelle Probleme der symmetrischen Kryptosysteme zu lösen:

- ⇝ das Schlüsselaustausch Problem
- ⇝ die digitale Signatur, die Verbindlichkeit realisiert

Wie bereits diskutiert, erfordert der Schlüsselaustausch bei der symmetrischen Verschlüsselung

1. entweder einen vor der eigentlichen verschlüsselten Nachrichtenübertragung über einen sicheren Kommunikationskanal ausgetauschten Key, den irgendjemand an die beiden Kommunikationsparteien verteilt hat
2. oder den Einsatz eines Key Distibution Centers

Beide Methoden des Schlüsselmanagements haben Nachteile.

Das zweite Problem der konventionellen, symmetrischen Kryptographie ist die digitale Signatur. Wenn der Einsatz der Kryptographie weite Verbreitung finden sollte – also nicht ausschließlich für militärische sondern auch für private und kommerzielle Zwecke – dann erfordern elektronische Nachrichten und Dokumente ein zu den klassischen Unterschriften auf Papierdokumenten gleichwertiges Verfahren.

Das jahrtausende alte Bild der Kryptographie (siehe KAHN, [118]) änderte sich radikal im Jahre 1976, als **Whitfield Diffie** und **Martin E. Hellman** — nach bedeutenden Vorarbeiten von **Ralph C. Merkle** — einen überraschenden Durchbruch bei dem Versuch erzielten, das Schlüsselmanagementproblem und das Problem der digitalen Signatur zu lösen [64].[1] Der Lösungsansatz von DIFFIE und HELLMAN unterschied sich radikal von allen anderen Versuchen. Das zentrale Konzept der Idee von DIFFIE und HELLMAN ist die Verwendung einer sogenannten One–Way Funktion (Einwegfunktion) zur Verschlüsselung.[2]

Die meisten Anwendungsgebiete, bei denen Public–Key Kryptosysteme eingesetzt werden, sind:

1. **Vertrauliche Nachrichtenübermittelung** (Privacy)
   Die vertrauliche Übermittelung von Nachrichten, so dass nur autorisierte Personen die Nachricht lesen können, ist natürlich die grundlegende Aufgabe jedes Kryptosystems.

2. **Authentifizierung**
   Unter dem Begriff Authentifizierung fasst man folgende drei Gebiete zusammen:

   - ☞ die Verifizierbarkeit des Absenders einer Nachricht. Authentifizierungsverfahren verwenden dazu sogenannte **Hash Funktionen** und **digitale Signaturen**.
   - ☞ **Passwort** und **Identifikationssysteme** zählt man auch zur Authentifizierung. Diese Systeme beweisen die Autorisierung, dass eine Person Zugriff auf Daten oder eine Einrichtung hat oder beweisen, dass jemand auch wirklich derjenige ist, der er behauptet zu sein.
   - ☞ **Verbindlichkeit**; diese Eigenschaft eines Public-Key Kryptosystems schützt gegenüber Personen, die vorgeben, etwas nicht behauptet zu haben, obwohl sie es tatsächlich behauptet haben.

3. **Key exchange** (Schlüsselaustausch)
   Public Key Kryptoverfahren werden häufig in Hybridsystemen für den Austausch des Schlüssels eingesetzt. Mit Hilfe des mit einem Public–Key Verfahrens übertragenen Schlüssels werden mittels symmetrischer Kryptosysteme die eigentlichen Nutzdaten verschlüsselt.

[1] In dem empfehlenswerten Buch von STEVEN LEVY [143] ist die Arbeit dieser Kryptographen sehr gut beschrieben.

[2] Mittlerweile ist bekannt (siehe zum Beispiel [32, p. 153]), dass einige Jahre vor der Publikation von DIFFIE und HELLMAN eine Reihe der in [64] und [182] entwickelten Ideen von JAMES ELLIS, CLIFFORD COCKS und MALCOLM WILLIAMSON vom britischen Geheimdienst **GCHQ** (Government Communications Headquarters) entwickelt wurden. Da diese Entwicklungen per Definition natürlich geheim bleiben mussten, bewirkte erst die Veröffentlichung von DIFFIE und HELLMAN einen rasanten Anstieg der Forschung in diesem Gebiet. Eine ausführliche Diskussion dieser Entwicklung in der GCHQ findet man in dem Buch von SIMON SINGH [204, Kap. 6, 388 ff.].

4. **Coin Flip**
   Das Szenario *Coin Flip* oder *Münzwurf* nennt man auch *bit commitment* und findet dann Anwendung, wenn beispielsweise Alice einen Schachpartner Bob in einer anderen Stadt hat und Alice und Bob möchten über Telefon oder E-Mail festlegen, wer bei der nächsten Schachpartie weiss spielt.

5. **Secret sharing**
   In diesem Szenario soll eine Cruise Missile gestartet werden. Dies ist aber nur dann möglich, wenn sich k Geheimnisträger zum gleichen Zeitpunkt treffen. Jeder der k Geheinmnisträger besitzt nun einen — nur ihm bekannten — Schlüssel. Nur wenn alle k Schlüssel gleichzeitig in das System eingeschoben werden, kann die Rakete gestartet werden. Wichtig ist zu sehen, dass dies nicht funktioniert, wenn nur $k-1$ Schlüssel vorhanden sind.

6. **Zero knowledge proof**
   Unter *zero knowledge proof* versteht man Szenarien, in denen man jemanden überzeugen will, dass man ein Problem gelöst hat, die Lösung selbst aber auf keinen Fall preisgeben möchte.

Diese unterschiedlichen Aufgaben, die Public–Key Systeme erfüllen können, werden über verschiedene **kryptographische Protokolle** durchgeführt.

6.2 Public–Key Kryptosysteme

Public–key Algorithmen basieren auf einem Schlüssel für die Chiffrierung und einen zweiten Schlüssel für die Dechiffrierung. Beide Keys hängen auf mathematisch subtile Weise voneinander ab.

6.2.1 Allgemeine Eigenschaften von Public–Key Verfahren

Die Public–Key Algorithmen können folgendermaßen charakterisiert werden:

> **Es ist mit keinem vertretbaren Rechenaufwand möglich, den Dechiffrierungs-Schlüssel allein aus der Kenntnis des Algorithmus und des Chiffrierungs–Schlüssels abzuleiten.**

Hier ist eine formale Definition eines Public–Key Kryptosystems:

Definition [6.11]:

Ein Public-Key Kryptosystem ist ein Paar

$$\{E_K\}_{K\in\{K\}} \text{ und } \{D_K\}_{K\in\{K\}}$$

von Algorithmen, die invertierbare Abbildungen darstellen:

$$E_K : \{M\} \longrightarrow \{M\}$$
$$D_K : \{M\} \longrightarrow \{M\}$$

auf einem endlichen Nachrichtenraum $\{M\}$, so dass:

1. für jedes $K \in \{K\}$, $\{E_K\}$ ist das Inverse von $\{D_K\}$,
2. für jedes $K \in \{K\}$ und $M \in \{M\}$, sind die Algorithmen $\{E_K\}$ und $\{D_K\}$ leicht zu berechnen,
3. für fast jeden Schlüssel $K \in \{K\}$ each easily computed algorithm equivalent to $\{D_K\}$ is computationally infeasible to derive from $\{E_K\}$
4. für jeden Schlüssel $K \in \{K\}$ gilt: Es ist mit geringem Rechenaufwand möglich, inverse Paare $\{E_K\}$ und $\{D_K\}$ aus K zu erzeugen.

In der Abbildung [6.2] ist den Ablauf der Verschlüsselung mit einem Public–Key Verfahren dargestellt. Die wesentlichen Schritte dabei sind:

1. Jedes End–System eines Netzwerkes erzeugt ein Schlüsselpaar, das für die Ver– und Entschlüsselung von zu sendenden bzw. empfangenen Nachrichten verwendet wird.

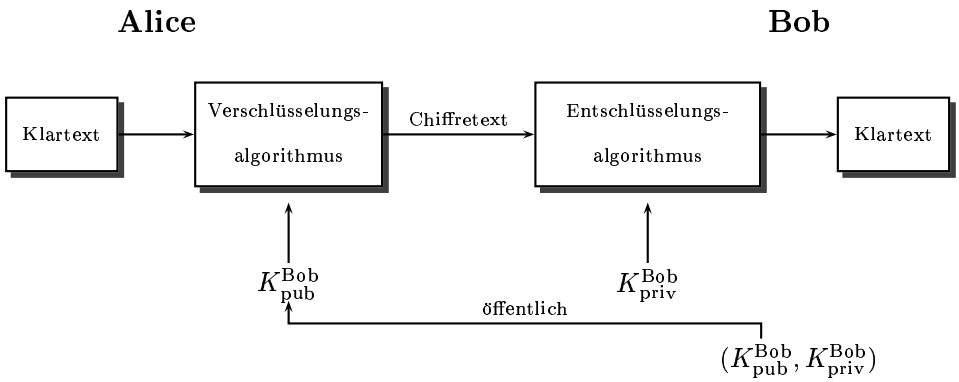


Abbildung 6.2: Modell des Public-Key Verfahrens.

2. Jedes System publiziert den Schlüssel, der zur Chiffrierung von Nachrichten *an das Endsystem* benutzt wird in einem öffentlich zugänglichen Ort, z.B. eine Datei. Dies ist der **Public Key** oder auch *öffentliche Schlüssel*. Der passende zweite Schlüssel — dies ist der **Private Key** oder *private Schlüssel* — bleibt geheim und ist damit nur dem erzeugenden Teilnehmer der Kommunikation bekannt.

3. Falls Alice eine chiffrierte Nachricht an Bob senden will, verschlüsselt sie diese mit Hilfe des Public Keys von Bob.

4. Wenn Bob die von Alice verschlüsselte Nachricht erhält, dechiffriert er diese mit seinem geheimen Private Key. Kein anderer Empfänger der chiffrierten Nachricht kann diese entschlüsseln, da nur Bob im Besitz des passenden Private Keys ist. Dies gewährleistet die Eigenschaft *privacy*.

Bei diesem Szenario haben also alle an der Kommunikation teilhabenden Parteien Zugriff auf die Public Keys. Die Private Keys werden von jedem Teilnehmer lokal generiert und müssen daher niemals verteilt werden. Solange ein System die Kontrolle über den privaten Schlüssel hat, ist die einlaufende Kommunikation sicher. Es ist für jeden Kommikationsteilnehmer jederzeit möglich, den privaten Schlüssel zu ändern, dabei muß der passende öffentlichen Schlüssel ebenfalls geändert und veröffentlicht werden.

Die Tabelle [6.1] stellt einige wichtige Aspekte der symmetrischen und asymmetrischen Verschlüsselungen gegenüber.

Sehen wir uns die wesentlichen Schritte eines Public-Key Verschlüsselungsverfahrens etwas genauer an. Wir legen dabei die Abbildung [6.3] zugrunde. Es gibt eine Nachrichtenquelle – diese nennen wir wieder Alice – die Nachrichten im Klartext generiert, $\mathtt{X} = [\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_M]$. Die M Elemente von $\mathtt{X}$ sind Buchstaben in irgendeinem endlichen Alphabet. Die Nachricht ist für den Empfänger Bob bestimmt. Bob erzeugt ein Schlüsselpaar:

- einen public key K^B_{pub}

- einen passenden private key K^B_{priv}

Der private Schlüssel K^B_{priv} ist ausschließlich Bob bekannt. Der öffentliche Schlüssel K^B_{pub} steht jedem zur Verfügung und kann daher von Alice benutzt werden.

Symmetrische Verschlüsselung	**Asymmetrische Verschlüsselung**
Notwendige Kriterien	*Notwendige Kriterien*
1. Der gleiche Algorithmus mit dem gleichen Schlüssel wird für die Ver– und Entschlüsselung eingesetzt. 2. Sender und Empfänger müssen den gleichen Algorithmus und den gleichen Schlüssel verwenden.	1. Ein Algorithmus wird benötigt für die Ver– und Entschlüsselung sowie ein Schlüsselpaar. Einer dieser Schlüssel wird für die Chiffrierung verwendet, der andere für die Dechiffrierung. 2. Sender und Empfänger müssen je einen der zueinander passenden Schlüssel besitzen.
Sicherheitsaspekte	*Sicherheitsaspekte*
1. Der Schlüssel muß geheim gehalten werden. 2. Es muß unmöglich sein – oder zumindest nicht praktikabel sein – eine Nachricht zu entschlüsseln, wenn keine andere Information als die verschlüsselte Nachricht zur Verfügung steht. 3. Die Kenntnis des Algorithmus sowie eine Reihe von Chiffraten darf nicht dazu führen, dass der Schlüssel abgeleitet werden kann.	1. Einer der beiden Schlüssel muß geheim gehalten werden. 2. Es muß unmöglich sein – oder zumindest nicht praktikabel sein – eine Nachricht zu entschlüsseln, wenn keine andere Information als die verschlüsselte Nachricht zur Verfügung steht. 3. Die Kenntnis des Algorithmus, die Kenntnis eines Schlüssels sowie die Kenntnis einer Reihe von Chiffraten darf nicht dazu führen, dass der andere Schlüssel abgeleitet werden kann.

Tabelle 6.1: Grundlegende Eigenschaften der symmetrischen und asymmetrischen Verschlüsselung.

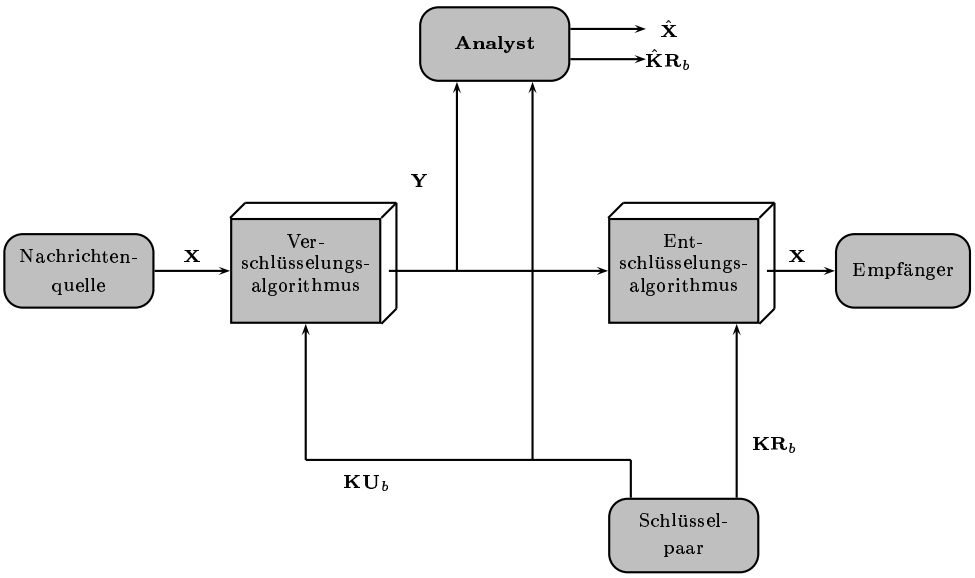


Abbildung 6.3: Public–Key Kryptosystem.

Mit der Nachricht $\mathtt{X}$ und Bobs öffentlichem Schlüssel K^B_{priv} als Input des Verschlüsselungsalgorithmus $\mathtt{E}$ erzeugt Alice den Chiffretext $\mathtt{Y} = [\mathtt{Y}_1, \mathtt{Y}_2, \ldots, \mathtt{Y}_N]$.

$$\mathrm{Y} = \mathtt{E}_{\mathrm{K}^B_{pub}}(\mathrm{X}).$$

Der gewünschte Empfänger Bob — nur dieser ist im Besitz des passenden private Keys — ist in der Lage, die von Alice durchgeführte Abbildung zu invertieren:

$$\mathrm{X} = \mathtt{D}_{\mathrm{K}^B_{priv}}(\mathrm{Y}).$$

Dabei wird der Chiffretext $\mathtt{Y}$ und Bobs private Key K^B_{priv} als Input des Dechiffrieralgorithmus $\mathtt{D}$ verwendet.

Ein Angreifer, der den Chiffretext $\mathtt{Y}$ abfängt und wie Alice ebenfalls Zugriff auf den öffentlichen Schlüssel K^B_{pub} von Bob hat, aber keine Kenntnis über den privaten Schlüssel K^B_{priv} oder den Klartext $\mathtt{X}$ verfügt, muss versuchen $\mathtt{X}$ zu rekonstruieren und/oder den privaten Schlüssel K^B_{priv} zu erhalten. Gemäß dem Kerckhoffs Prinzip muß dabei davon ausgegangen werden, dass der Angreifer Kenntnis hat über den Ver– bzw. Entschlüsselungsalgorithmus. Falls der Angreifer nur an der aktuellen Nachricht interessiert ist, dann ist das Ziel seiner Kryptoanalyse den Klartext $\mathtt{X}$ durch eine Schätzung $\hat{\mathrm{X}}$ zu rekonstruieren. In den meisten Fällen sind Angreifer jedoch daran interessiert, auch zukünftige Nachrichten zu lesen, in diesem Fall wird versucht, den privaten Schlüssel K^B_{priv} zu erhalten, indem eine erste Approximation $\mathrm{K}^{\hat{B}}_{priv}$ erstellt wird.

Wie erwähnt, kann einer der beiden Schlüssel für die Verschlüsselung einer Nachricht, der andere Key für die Entschlüsselung verwendet werden. Diese Eigenschaft von Public–Key Verfahren ermöglicht die Implementierung eines völlig

anderen Kryptosystems. Das Szenario in Abbildung [6.3] liefert die grundlegende Eigenschaft *Vertraulichkeit*, weil durch die Anwendung dieses Verfahrens eine Nachricht auf sichere Weise von Alice zu Bob übertragen werden kann, so dass nur autorisierte Personen den Inhalt der Nachricht lesen können.

Wie aus der Abbildung [6.4] hervorgeht, ermöglichen Public–Key Verfahren auch die Umsetzung der **Authentifikation**.

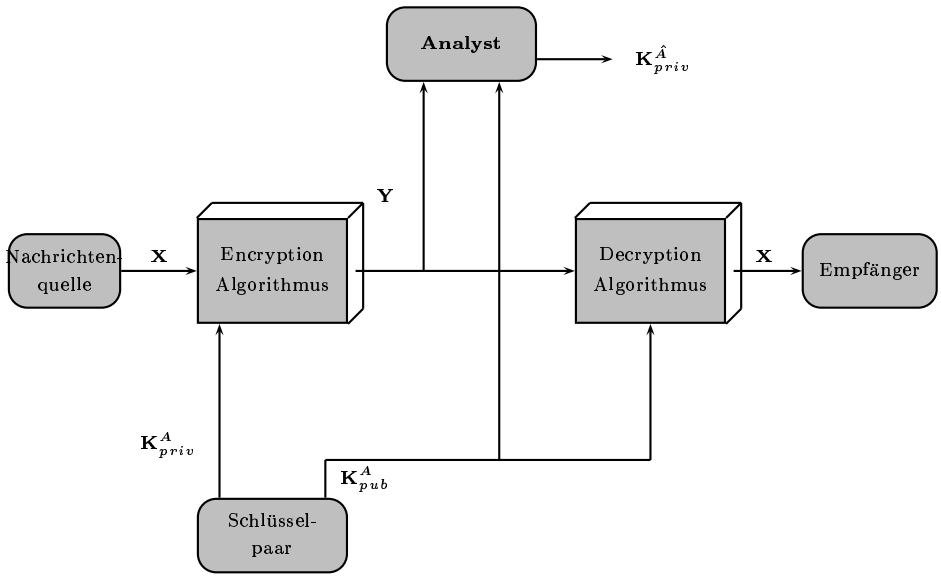


Abbildung 6.4: Prinzip der Public–Key Authentifikation.

Das Schema zur Authentifizierung lautet:

$$\mathrm{Y} = \mathrm{E}_{\mathrm{K}^{A}_{priv}}(\mathrm{X}),$$
$$\mathrm{X} = \mathrm{D}_{\mathrm{K}^{A}_{pub}}(\mathrm{Y}).$$

Soll das Verfahren der Authentifizierung mittels Public–Key Verfahren umgesetzt werden, sind folgende Schritte durchzuführen:

❶ *Szenario:* Alice will Bob eine signierte Nachricht zukommen lassen.

❷ Alice erstellt die zu signierende Message und verschlüsselt diese mit ihrem eigenen privaten Schlüssel.[3]

❸ Alice sendet das Chiffrat an Bob.

❹ Bob besorgt sich den öffentlichen Schlüssel von Alice.

[3] Um präzise zu sein, bei diesem Schritt handelt es sich nicht um eine echte Chiffrierung, da ein Angreifer mit Alices Public-Key dieses Chiffrat problemlos entziffern kann. Der Punkt ist, dass Alice den Public-Key Verschlüsselungsalgorithmus auf den Klartext anwendet und dabei ihren privaten Schlüssel als Input benutzt.

❺ Bob wendet den Entschlüsselungsalgorithmus des Public-Key Verfahrens an mit Input:

(a) empfangenes Chiffrat von Alice

(b) Public–Key von Alice

❻ Falls bei dieser Operation Klartext entsteht, muss die Nachricht von Alice stammen — sie hat daher unbestreitbar die Nachricht verfaßt — da nur sie im Besitz des passenden privaten Schlüssels ist. Daher dient die komplette verschlüsselte Nachricht als digitale Signatur. Eine zusätzliche Eigenschaft, die dieses Verfahren liefert besteht in der Beobachtung, dass ohne Zugriff auf Alice privaten Schlüssel keine Möglichkeit besteht, während der Übertragung den Inhalt der Nachricht zu ändern. Damit gewährleistet dieses Verfahren auch die **Integrität**.

In dem oben dargestellten Verfahren wird die komplette Nachricht, die Alice an Bob sendet, verschlüsselt. Dies erfordert — obwohl Authentifikation und Integrität umgesetzt werden — einerseits einen hohen Rechenaufwand, andererseits viel Speicherplatz. Für praktische Zwecke muß jedes Dokument im Klartext erhalten bleiben. Weiterhin muß eine verschlüsselte Kopie dieses Klartexts ebenfalls gespeichert werden, so dass im Fall einer Streitigkeit das Original zur Verfügung steht. Eine effizientere Weise, das gleiche Ergebnis zu erzielen besteht darin, einen kleinen Block von Bits zu verschlüsseln, der eine Funktion der zu versendenden Nachricht ist. Dieser Block — diesen nennt man auch **Authentikator** oder **Message Digest** — muss die Eigenschaft haben, dass es nicht möglich ist, das Dokument zu ändern ohne den Authentikator zu ändern.

Es ist wichtig zu realisieren, dass das oben beschriebene Verfahren keine Vertraulichkeit gewährleisten kann. Das bedeutet, die versendete Nachricht ist sicher gegenüber nicht–authorisierte Änderung aber nicht gegenüber Abhören. Dies ist offensichtlich, wenn für die Authentifikation ein Authentikator eingesetzt wird, da dann die Nachricht im Klartext versendet wird.

Es ist jedoch möglich, sowohl Vertraulichkeit als auch Authentifikation — und Integrität — zu gewährleisten, wenn das Public–Key Verfahren zweifach angewendet wird:

$$\begin{aligned} Z &= E_{K^B_{pub}}[E_{K^A_{priv}}(X)] && \text{Chiffrierung,} \\ X &= D_{K^A_{pub}}[D_{K^B_{priv}}(Z)] && \text{Dechiffrierung.} \end{aligned}$$

Hier sind also folgende Schritte erforderlich:

① Die Nachricht wird mit dem privaten Schlüssel des Senders chiffriert. Dieser Schritt gewährleistet Authentifikation.

② Der Sender verschlüsselt die Nachricht ein zweites Mal, diesmal wird der öffentliche Schlüssel des Empfängers als Input benutzt. Dieser Schritt realisiert die Vertraulichkeit, da nur der beabsichtigte Empfänger — der allein im Besitz des passenden privaten Schlüssels ist — die Nachricht dechiffrieren kann.

Der Nachteil dieses Verfahrens liegt darin, dass in jeder Kommunikation der Public–Key Algorithmus — der in der Regel sehr komplex und rechenaufwändig ist — viermal anstatt zweimal angewendet werden muss (siehe Abbildung [6.5]).

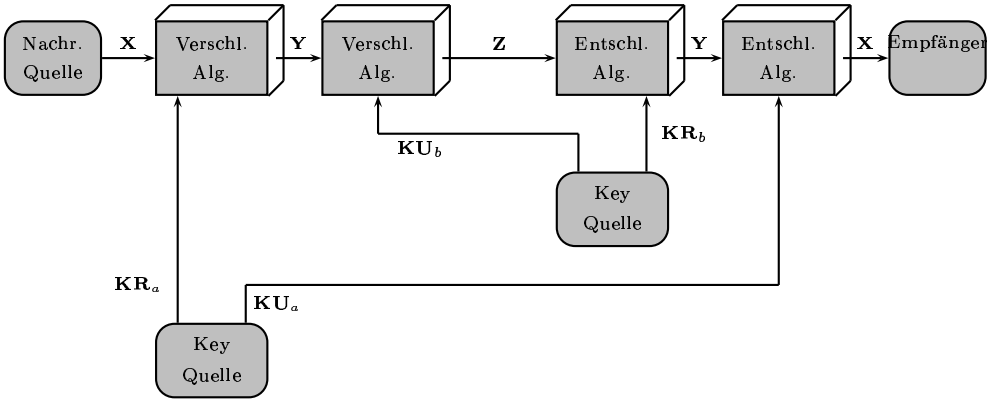


Abbildung 6.5: Kombination von Vertraulichkeit und Authentifikation mit Public–Key Kryptosysteme.

6.2.2 Anwendungen von Public–Key Kryptosystemen

Es ist wichtig an diesem Punkt, ein Aspekt der Public-Key Kryptographie klar herauszuarbeiten, der ansonsten zu Verwirrung führt. Public-Key Kryptosysteme sind dadurch charakterisiert, dass ein kryptographischer Algorithmus mit zwei Schlüsseln eingesetzt wird, einer der Schlüssel wird geheim gehalten, der andere wird öffentlich verfügbar gemacht. Abhängig von der Zielsetzung benutzt der Sender einer Nachricht entweder seinen eigenen privaten Schlüssel oder den Public Key des Senders — oder beide — um eine kryptographische Funktion durchzuführen. Grob kann die Verwendung von Public–Key Kryptosysteme in drei Kategorien eingeteilt werden:

- **Ver– und Entschlüsselung**
  Der Sender einer Nachricht verschlüsselt diese mit den Public Key des Empfängers.

- **Digitale Signatur**
  Der Sender signiert eine Nachricht mit seinem privaten Schlüssel. Die Signatur wird durch eine kryptographischen Algorithmus erstellt, der entweder auf eine Nachricht oder einen kleinen Block angewendet wird, der wiederum eine Funktion der Nachricht ist,

- **Schlüsselaustausch**
  Zwei Parteien erarbeiten zusammen einen Session Key. Hier sind verschiedene Zugänge möglich, die den/die privaten Schlüssel von einem Teilnehmer oder beiden verwenden.

Einige Algorithmen können alle drei Anwendungen umsetzen, andere Verfahren können nur für eine oder zwei Anwendungen eingestzt werden. Die Tabelle [6.2] zeigt die Anwendungen für verschiedene Public-Key Algorithmen.

Algorithmus	Encr./Decryption	Digital Sign.	Key Exchange
RSA	Ja	Ja	Ja
Diffie–Hellman	Nein	Nein	Ja
Elliptische Kurven	Ja	Ja	Ja
DSS	Nein	Ja	Nein

Tabelle 6.2: Anwendungsbereiche für Public–Key Kryptosysteme.

6.2.3 Anforderungen an Public–Key Kryptosysteme

Die oben vorgestellten Kryptosysteme hängen von einem kryptographischen Algorithmus ab, der auf zwei Schlüsseln basiert, die auf subtile Weise voneinander abhängen. DIFFIE und HELLMAN postulierten solch ein System in ihrer Arbeit [64] ohne jedoch zeigen zu können, dass solch ein Algorithmus auch tatsächlich existiert. Sie waren jedoch in der Lage, die Bedingungen herauszuarbeiten, die solch ein Algorithmus erfüllen muss.

- Es ist vom Rechenaufwand leicht für Bob, ein Schlüsselpaar zu erzeugen, *i.e.* den öffentlichen Schlüssel K_{pub}^{B} und den zugehörigen privaten Schlüssel K_{priv}^{B}.
- Es ist vom Rechenaufwand einfach für den Sender Alice, die den öffentlichen Schlüssel und den zu verschlüsselnden Klartext kennt, den entsprechenden Chiffretext C zu generieren.

$$\mathrm{C} = \mathrm{E}_{\mathrm{K}_{pub}^{B}}(M)$$

- Es ist vom Rechenaufwand einfach für den Empfänger Bob den erhaltenen Chiffretext mit Hilfe seines privaten Schlüssels zu dechiffrieren um die ursprüngliche Nachricht zurück zu gewinnen.

$$\mathrm{M} = \mathrm{D}_{\mathrm{K}_{priv}^{B}}(\mathrm{C}) = \mathrm{D}_{\mathrm{K}_{priv}^{B}}[\mathrm{E}_{\mathrm{K}_{pub}^{B}}(\mathrm{M})]$$

- Es ist für einen Angreifer rechnerisch undurchführbar, aus der Kenntnis des Public-Keys K_{pub}^{B} von Bob den Private Key von Bob K_{priv}^{B} abzuleiten.
- Es ist für einen Angreifer rechnerisch undurchführbar, aus der Kenntnis des Public-Keys K_{pub}^{B} und eines Chiffretexts den Private Key K_{priv}^{B} abzuleiten.

- Die Verschlüsselungs- und Entschlüsselungsfunktionen können in beliebiger Reihenfolge angewendet werden:

$$M = E_{K^B_{pub}}[D_{K^B_{priv}}(M)]$$
$$M = D_{K^B_{pub}}[E_{K^B_{priv}}(M)]$$

Die letzte Bedingung ist vielfach sehr zweckmäßig jedoch nicht notwendig für alle Public-Key Anwendungen. Das ElGamal Verfahren hat diese Eigenschaft beispielsweise nicht.

Betrachten wir diese Anforderungen etwas detaillierter. Die oben aufgeführten Diffie–Hellman Anforderungen lassen sich auf die Notwendigkeit einer sogenannten **trap–door Einwegfunktion** zusammenfassen. Eine **Einwegfunktion** bildet eine Menge in eine andere Menge ab, so dass jeder Funktionswert ein eindeutiges Urbild hat. Zusätzlich gilt die Randbedingung, dass die Berechnung des Funktionswertes einfach ist, die Umkehrung jedoch undurchführbar.

$$y = f(x) \qquad \text{einfach,}$$
$$x = f^{-1}(y) \qquad \text{undurchführbar.}$$

Beispiel [6.13]

Das Zerschlagen einer Glasplatte in Tausende Bruchstücke ist einfach, das Zusammensetzen sämtlicher Glasscherben zu einer Glasplatte ist ein hartes Problem – i.e. in der Praxis undurchführbar.

Was bedeutet eigentlich konkret die Aussage *vom Rechenaufwand einfach* bzw. *undurchführbar*? Wenn man diese Frage diskutiert, betritt man unweigerlich die Sphäre der **Komplexitätstheorie**.[4] In diesem Zusammenhang bedeutet *einfach*, dass ein Problem in polynomialer Zeit als Funktion der Eingabelänge gelöst werden kann. Das bedeutet, wenn die Länge des Inputs (also z.B. ein Schlüssel) n Bits beträgt, dann ist die Laufzeit um die Funktion zu berechnen proportional zu $n^a, a = const$. Solche Algorithmen gehören zur Klasse $\mathcal{P}$.

Der Ausdruck *undurchführbar* ist wesentlich schwieriger zu handhaben. Generell gilt die Aussage, ein Problem nennt man *undurchführbar*, falls die Lösung in der Zeit schneller als polynomial wächst mit der Länge des Inputs. Beträgt beispielsweise die Inputlänge n Bit und die Zeit, diese Funktion zu berechnen, ist proportional zu 2^n, dann wird das Problem als *undurchführbar* betrachtet. Die Laufzeit des Algorithmus wächst also *exponentiell* mit der Länge des Inputs.

Unglücklicherweise ist es schwierig zu bestimmen, ob ein gegebener Algorithmus diese Eigenschaft hat. Weiterhin konzentrieren sich Komplexitätsbetrachtungen

[4] Für eine exzellente Einführung in die Komplexitätstheorie siehe [91]. Obwohl dieses Buch nicht mehr den aktuellen Stand der Dinge enthält ist es lohnenswert, es als 'Klassiker' zu lesen. Eine sehr empfehlenswerte nicht–technische Einführung in die Komplexitätstheorie findet man in [103].

von Algorithmen stets auf den sogenannten **worst case**. Das bedeutet, dass ein Problem in die Klasse *undurchführbar* eingeordnet wird, es aber durchaus viele spezielle Fälle (sogenannte Instanzen) gibt, bei denen die Lösung in polynomialer Laufzeit vorliegt. Dies ist nicht zweckmäßig für kryptologische Methoden. Schließlich muss man sich stets der Tatsache bewußt sein, dass es im allgemeinen nicht bekannt ist, ob ein Problem, das als undurchführbar betrachtet wird, nicht doch in polynomialer Laufzeit gelöst werden kann. Das bedeutet also, für viele — auch in der Praxis sehr relevante Probleme — sind nur Lösungen — *i.e.* Algorithmen — bekannt mit exponentieller Laufzeit. Es ist dabei mathematisch weder bewiesen noch widerlegt, ob ein Polynomialzeit–Algorithmus zur Lösung dieses Problems überhaupt existiert.

Eine **Trap–door Einwegfunktion** ist vom Rechenaufwand einfach in einer Richtung zu berechnen und mit vertretbarem Aufwand nicht zu invertieren, *es sei denn, zusätzliche Informationen sind bekannt.* Mit Hilfe dieser Zusatzinformation kann die Umkehrung dann in polynomialer Laufzeit berechnet werden. Zusammenfassend kann also konstatiert werden:

> Eine Trap–door Einwegfunktion ist eine Familie invertierbarer Funktionen f_k, so dass:
>
> | | |
> |---|---|
> | $y = f_k(x)$ | einfach, falls k und x bekannt |
> | $x = f_k^{-1}(y)$ | einfach, falls k und y bekannt |
> | $x = f_k^{-1}(y)$ | undurchführbar, falls y bekannt aber k unbekannt |

Die Entwicklung eines praktisch nutzbaren Public–Key Verfahrens hängt entscheidend davon ab, ob man eine geeignete Trap–door Einwegfunktion findet.

Beispiel [6.14]

Ein einfaches Beispiel einer One–Way Funktion ist die Berechnung eines Produkts der elf Polynome $(x-1)$ bis $(x-11)$. Die Berechnung dieses Produkts ist durch einen (halbwegs begabten) Mittelstufenschüler mit etwas Mühe machbar:

$$\begin{aligned}\prod_{r=1}^{11}(x-r) =& x^{11} - 66x^{10} + 1925x^9 - 32670x^8 + 357423x^7 - 2637558x^6 \\ &+ 13339535x^5 - 45995730x^4 + 105258076x^3 \\ &- 150917976x^2 + 120543840x - 39916800.\end{aligned}$$

Ist jetzt umgekehrt das obige Poynom 11. Grades gegeben und man hat die Aufgabe, die Nullstellen dieses Polynoms zu finden, dann ist dies nicht möglich.

6.2.4 Beispiel einer Trap-door Funktion

Die typischen Eigenschaften einer Trap–door Funktion kann anhand des folgenden **Subset Problem**s ([106]) gut illustriert werden. Wir betrachten dazu einen Hohlzylinder der Höhe H. Dieser Hohlzylinder muss mit einer Anzahl massiver Zylinder gefüllt werden, die unterschiedliche Höhen haben. Wir bezeichnen die verschiedenen Höhen der gegebenen n Zylinder mit $h_1, h_2, \ldots, h_n$. Das Subset - Problem besteht darin, herauszufinden, welche Kombination der n massiven Zylinder in der Höhe genau in den Hohlzylinder der Höhe H past. Dieses Problem ist äquivalent dazu, eine bestimmte Teilmenge aus einer Menge von n Zahlen herauszusuchen, so dass die Summe der Zahlen dieser Teilmenge genau die gegebene Zahl H ist.

Ein Verschlüsselungsverfahren, das auf diesem Problem basiert, funktioniert folgendermaßen. Der Sender einer Nachricht codiert seine Message als binären Bitstring, z.B. eine 5-Bit Codierung. Damit können $2^5 = 32$ Symbole dargestellt werden, e.g. A = 00000; B=00001, C=00010 etc. Dann besorgt sich der Sender der Nachricht den Public-Key des Empfängers – beispielsweise aus einem öffentlichen Verzeichnis. Dieser Key ist eine Menge von n Zahlen $a = (a_1, a_2, \ldots, a_n)$, diese Menge nennt man *Code-Vector*.

Die Verschlüsselung der Nachricht funktioniert folgendermaßen. Der Sender teilt den binären String in eine Anzahl von Blöcken der Länge von n Bits. Jeder Block kann als n-dimensionaler Vektor $x = (x_1, x_2, \ldots, x_n); x_i \in \{0, 1\}$ betrachtet werden. Dann berechnet der Sender das Skalarprodukt jedes Blocks mit dem Code-Vektor $x \cdot a$. Die resultierende Summe

$$C = \sum_{l=1}^{n} x_l \cdot a_l$$

stellt den Chiffretext dar, der über einen unsicheren Kommunikationskanal an den Empfänger gesendet wird.

Ein Angreifer, der diese Kommunikation belauscht und der begierig darauf ist, die Nachricht zu entschlüsseln, ist mit folgendem Problem konfrontiert: Gegeben ist der Chiffretext und die Zahlen $a_1, a_2, \ldots, a_n$. Wie lautet der Klartext? Da die x_i entweder `0` oder `1` sind, ist sein Problem gleichbedeutend dazu, aus der Menge der n Zahlen $a_1, a_2, \ldots, a_n$ die Teilmenge herauszufinden, die sich zu C aufaddiert. Der legitime Empfänger der Nachricht ist natürlich mit dem gleichen Problem konfrontiert. Er verfügt jedoch über zusätzliche Informationen: Den Dechiffrierungsschlüssel und einen geheimen trapdoor Parameter.

Wir wenden dieses Schema an, um den Klartext `how do you do` zu verschlüsseln Mit Hilfe des 5-Bit Codes aus der Tabelle [6.3], kann dieser Klartext in einen binären String transformiert werden.

Damit lauten die ersten 20 Bits:

```
00111 01110 10110 11010
```

wobei der letzte Block für ein Leerzeichen steht. Wir wählen den folgenden

A = 00000	B = 00001	C = 00010	D = 00011	E = 00100
F = 00101	G = 00110	H = 00111	I = 01000	J = 01001
K = 01010	L = 01011	M = 01100	N = 01101	O = 01110
P = 01111	Q = 10000	R = 10001	S = 10010	T = 10011
U = 10100	V = 10101	W = 10110	X = 10111	Y = 11000
Z = 11001	= 11010	, = 11011	? = 11100	; = 11101

Tabelle 6.3: Eine einfache 5-bit Codierung.

Codevektor:

$$a = (2292, 1089, 211, 1625, 1283, 599, 759, 315, 2597, 2463).$$

Damit:

$$\begin{aligned} a_1 &= 2292 \\ a_2 &= 1089 \\ a_3 &= 211 \\ a_4 &= 1625 \\ a_5 &= 1283 \\ a_6 &= 599 \\ a_7 &= 759 \\ a_8 &= 315 \\ a_9 &= 2597 \\ a_{10} &= 2463 \end{aligned}$$

Der erste Klartextblock lautet:

$$x = (0, 0, 1, 1, 1, 0, 1, 1, 1, 0)$$

Damit führt das Skalarprodukt des Codevektors mit den Informationsbits auf den folgenden Chiffretext:

$$\begin{aligned} C &= a \cdot x \\ &= \sum_{l=1}^{10} a_i \cdot x_i \\ &= a_1x_1 + a_2x_2 + \cdots a_{10}x_{10} \\ &= 2292 \cdot 0 + 1089 \cdot 0 + 211 \cdot 1 + 1625 \cdot 1 + 1283 \cdot 1 \\ &\quad + 599 \cdot 0 + 759 \cdot 1 + 315 \cdot 1 + 2597 \cdot 1 + 2463 \cdot 0 \\ &= 6790. \end{aligned}$$

Um diese Nachricht zu entschlüsseln, muss herausgefunden werden, welche der zehn Zahlen a_i sich zu 6790 aufaddieren. Für jedes a_i in der Summe ist das entsprechende x_i gleich 1, andernfalls 0.

Heute ist keine Lösung des Subset-Problems bekannt, die sich wesentlich von der Brute-Force Methode unterscheidet. Diese besteht darin, einfach alle möglichen

2^n Teilmengen der n Zahlen a_i zu bilden, alle Elemente jeder Teilmenge aufzuaddieren und prüfen, ob ihre Summe gleich dem Chiffretext C ist. In unserem Beispiel hat der Codevektor a die zehn Elemente $a_i, 1 \leq i \leq 10$. Daher ist es in diesem Fall einfach, den Chiffretext zu entschlüsseln indem man einfach alle $2^{10} = 1.024$ mögliche Kombinationen der öffentlich zugänglichen a_i's bildet um den Nachrichtenvektor x zu rekonstruieren. Aus diesem Grund folgern wir, dass die Anzahl der Komponenten des gewählten Codevektors zu klein ist um ausreichende Sicherheit zu gewährleisten.

Da das Subset-Problem ein typisches Problem der Klasse der $\mathcal{NP}$-vollständigen Probleme ist (siehe [91]), wächst die Laufzeit, dieses Problem zu lösen exponentiell mit der Länge des Inputs. Das bedeutet, wenn wir einen Codevektor mit 1.000 Komponenten a_i wählen, muß ein Angreifer 2^{1000} Teilmengen prüfen. In diesem Fall ist es vom Rechenaufwand nicht vertretbar, alle möglichen Teilmengen zu bilden.

Falls die Komponenten des Codevektors zufällig gewählt werden, ist nicht einmal der legitime Empfänger der Nachricht in der Lage, die x_i's aus dem Chiffrat C zu extrahieren. Falls jedoch die Komponenten des Codevektors eine bestimmte Struktur haben, dann reicht eine kleine zusätzliche Information aus, den Klartext aus dem Chiffretext C zu erhalten. Hierbei macht man sich zu Nutze, dass das Subset-Problem für bestimmte Codevektoren einfach zu lösen ist.

Das Verfahren funktioniert folgendermaßen:

Wir nehmen solch einen speziellen Codevektor – wir nennen diesen a' – und transformieren diesen in einen neuen Vektor a. Dieser Vektor a wird in einem öffentlichen Verzeichnis als Public Key publiziert. Mit einer zusätzlichen, geheimen Information ist der Empfänger einer verschlüsselten Nachricht in der Lage, das nicht berechenbare Teilsummen Problem, das auf dem Vektor a basiert, in ein einfacheres, äquivalentes Problem, basierend auf dem Vektor a' umzuwandeln.

Um den Schlüssel a zu konstruieren, wählt die Person, die eine verschlüsselte Nachricht erhalten möchte einen Codevektor

$$a' = (a'_1, a'_2, \ldots, a'_n)$$

Dabei ist jedes Element a'_i dieses Vektors größer als die Summe der vorhergehenden Terme:

$$a'_i > \sum_{k=1}^{i-1} a'_k$$

Diese Eigenschaft trifft beispielsweise auf den folgenden Vektor zu:

$$a' = (3, 5, 11, 20, 41, 83, 169, 340, 679, 1358)$$

Das Skalarprodukt dieses speziellen Codevektors mit einem binären Klartextvektor $x = (x_1, x_2, \ldots, x_n)$ liefert

$$\begin{aligned} C' &= a' \cdot x \\ &= 3 \cdot x_1 + 5 \cdot x_2 + 11 \cdot x_3 + 20 \cdot x_4 + \\ &\quad + 41 \cdot x_5 + 83 \cdot x_6 + 169 \cdot x_7 + 340 \cdot x_8 + 679 \cdot x_9 + 1358 \cdot x_{10} \end{aligned}$$

Um zu sehen, dass diese Form der Verschlüsselung leicht zu invertieren ist, betrachten wir den Fall, dass dieses Skalarprodukt den Wert $C' = 1260$ ergibt.

In diesem Fall entspricht die Entschlüsselung ebenfalls dem Subset Problem, die Lösung x ist aber leicht zu erhalten. Zunächst ist klar, dass der Wert a'_{10} des Codevektors nicht in der Summe C' enthalten sein kann, da deises Element des Codevektors größer als C' ist. Daher muß der binäre Wert von x_{10} 0 sein. Das größte Element von a', das in die Summe C' passt, ist der Term a'_9. Da die Summe der acht verbleibenden Terme des Codevektors kleiner als 679 ist, muß das Element a'_9 Teil der Summe sein, andernfalls ist es unmöglich, alle Komponenten zu 1260 aufzuaddieren. Daher ist $x_9 = 1$. Daher kann nun die Gleichung $C' = a' \cdot x = 1260$ in folgende Form umgeschrieben werden:

$$\begin{aligned} C' &= a' \cdot x \\ &= 3 \cdot x_1 + 5 \cdot x_2 + 11 \cdot x_3 + 20 \cdot x_4 + \\ &\quad + 41 \cdot x_5 + 83 \cdot x_6 + 169 \cdot x_7 + 340 \cdot x_8 + 679 + 0 \\ &\stackrel{!}{=} 1260 \end{aligned}$$

Nun subtrahiert man 679 auf beiden Seiten der Gleichung:

$$\begin{aligned} C' &= a' \cdot x \\ &= 3 \cdot x_1 + 5 \cdot x_2 + 11 \cdot x_3 + 20 \cdot x_4 + \\ &\quad + 41 \cdot x_5 + 83 \cdot x_6 + 169 \cdot x_7 + 340 \cdot x_8 \\ &\stackrel{!}{=} 581 \end{aligned}$$

Daher reduziert sich das ursprüngliche Problem auf die Frage: Welche der verbleibenden acht Komponenten des Codevektors summieren sich zu dem Wert 581? Verfährt man auf die gleiche Wiese wie zuvor, dann ergibt sich der Nachrichtenvektor schließlich zu

$$x = (0, 0, 1, 1, 1, 0, 1, 1, 1, 0)$$

Das einzig verbleibende Problem besteht darin, wie man den Vektor a aus dem Codevektor a' erhält. Das Procedere besteht darin, zwei große Zahlen m und w zu wählen. Dann setzt man:

$$a_i \equiv a'_i \cdot w \bmod m$$

Da die Modulo Operation monoton wachsende oder fallende stetige Funktionen in unstetige Funktionen transformiert, treten die resultierenden a_is in zufälliger Ordnung auf, obwohl die a'_is monoton steigen. Ohne die Kenntnis der Zahlen m und w ist es nun für einen Angreifer nicht mehr möglich, den Klartext zu rekonstruieren.

Andererseits ist es für den legitimen Empfänger der Nachricht – dieser kennt die Werte von m und w, diese beiden Zahlen stellen den privaten Schlüssel dar – kein Problem, den Vektor a in den aufsteigend geordneten Vektor a' zurückzutransformieren. Dabei muß das multiplikative Inverse w^{-1} bestimmt werden mit

$$w \cdot w^{-1} \bmod m \equiv 1 \bmod m$$

Dies ist eine Aufgabe für den erweiterten EUKLIDischen Algorithmus, der in Kapitel [18.2] diskutiert wurde. Damit folgt:

$$\begin{aligned} Cw^{-1} \bmod m &= (\sum_i a_i\ x_i)\ w^{-1} \bmod m \\ &= (\sum_i (a'_i \cdot w \bmod m)\ x_i)\ w^{-1} \bmod m \\ &= (\sum_i a'_i \cdot x_i) w \cdot w^{-1} \bmod m \\ &= (\sum_i a'_i \cdot x_i) \\ &= C' \end{aligned}$$

Damit haben wir gezeigt, dass das Problem den Chiffretext C zu entschlüsseln äquivalent ist zu dem Problem, den Chiffretext C' zu entschlüsseln – vorausgesetzt die Zahlen m und w sind bekannt. Letzteres ist aber ein Problem, das in polynomialer Laufzeit gelöst werden kann.

6.3 Der RSA Algorithmus

Die bahnbrechende Arbeit von DIFFIE und HELLMAN ([64]) führte eine völlig neue Vorgehensweise in der Kryptologie ein. Gleichzeitig waren die Kryptologen herausgefordert, einen kryptologischen Algorithmus zu entwickeln, der die Anforderungen eines Public-Key Kryptosystems umsetzen kann. Mit die erste Antwort auf diese Herausforderung wurde von **Ron Rivest**, **Adi Shamir** und **Leo Adleman** am MIT entwickelt und im Jahre 1978 publiziert [182]. Das RIVEST-SHAMIR-ADLEMAN (RSA) Verfahren ist seit seiner Entstehung das meist verbreitete Public-Key Verfahren und generell akzeptiert als *der* Standard.

Abbildung 6.6: Ronald Rivest, Adi Shamir und Leonard Adleman.

Das RSA Verfahren ist eine Blockchiffre, in der der Klartext und Chiffretext eine ganze Zahl zwischen 0 und $n-1$ ist, n ist dabei eine beliebige Zahl. Der Klartext wird in Blöcken verschlüsselt, dabei hat jeder Block einen binären Wert kleiner als n. Damit muß die Blockgröße kleiner oder gleich $\log_2 n$ sein. In der Praxis ist die Blockgröße 2^k Bits, wobei $2^k < n < 2^{k+1}$. Die Ver– und Entschlüsselung hat die folgende Form (M – Klartextblock, C Chiffretextblock)

$$C = M^e \bmod n$$
$$M = C^d \bmod n = (M^e)^d \bmod n = M^{cd} \bmod n$$

Sender und Empfänger der Nachricht müssen den Wert von n kennen. Der Sender kennt den Wert e und ausschließlich dem Empfänger ist der Wert von d bekannt. Damit hat man ein Public-Key Kryptosystem vorliegen mit Public Key $\{e, n\}$ und Private Key $\{d, n\}$. Damit mit diesem Algorithmus ein zufriedenstellendes Public-Key Kryptosystem aufgebaut werden kann, müssen folgende Anforderungen erfüllt sein:

① Es ist möglich, Werte für e, d, n zu finden, so dass

$$M^{ed} = M \bmod n$$

für alle $M < n$.

② Es ist relativ leicht, M^e und C^d für alle Werte von $M < n$ zu berechnen.

③ Es ist vom Rechenaufwand praktisch undurchführbar, den Wert von d zu bestimmen, wenn e und n bekannt sind.

Wir betrachten nun zunächst die erste Anforderung etwas detaillierter. Gesucht ist also eine mathematische Beziehung der Form:

$$M^{ed} = M \bmod n$$

Ein Corollar des EULER Theorems, das in Kapitel [17] diskutiert ist, liefert die folgende Aussage: Gegeben sind

- zwei Primzahlen p, q
- zwei ganze Zahlen m und n mit $n = pq$ und $0 < m < n$
- und eine beliebige ganze Zahl k.

Dann gilt die folgende Beziehung:

$$m^{k\phi(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \bmod n$$

wobei $\phi(n)$ die EULERsche Totientenfunktion ist. Wir erhalten die gewünschte Eigenschaft, wenn gilt:

$$ed = k\phi(n) + 1$$

Dies ist äquivalent zu:

$$ed \equiv 1 \bmod \phi(n)$$

oder

$$d \equiv e^{-1} \bmod \phi(n)$$

Das bedeutet, dass e und d multiplikativ invers zueinander sind $\bmod \phi(n)$. Aufgrund der modularen Arithmetik gilt dies genau dann und nur dann, wenn d und e coprim zu $\phi(n)$ ist. Oder:

$$\mathtt{ggT}\ (\phi(n), d) = 1.$$

Zusammenfassend kann also festgestellt werden:

p, q zwei Primzahlen	(geheim, frei gewählt, chosen)
$n = pq$	(öffentlich, berechnet)
e, mit $\mathtt{ggT}(\phi(n), e) = 1; 0 < e < \phi(n)$	(öffentlich, frei gewählt)
$d \equiv e^{-1} \bmod \phi(n)$	(geheim, berechnet)

Der private Schlüssel besteht aus dem Paar $\mathrm{K}_{priv} = [d, n]$, der Public-Key ist das Paar $\mathrm{K}_{pub} = [e, n]$ oder umgekehrt.

Angenommen, Alice hat ihren Public-Key veröffentlicht und Bob will eine Nachricht M verschlüsselt an Alice senden. Aus dem gegebenen Klartext M – codiert als Zahl – und dem öffentlichen Schlüssel von Alice berechnet Bob

$$C = M^e \bmod n$$

und sendet den Chiffretext C – das ist wiederum eine Zahl – an Alice. Wenn Alice den Chiffretext erhält, entschlüsselt sie diesen durch Berechnung von

$$M = C^d \bmod n$$

Da nur sie im Besitz des zu $\mathrm{K}_{pub} = [e, n]$ passenden privaten Schlüssels $\mathrm{K}_{priv} = [d, n]$ ist, ist nur sie in der Lage, den Chiffretext zu entschlüsseln. Diese Eigenschaft gewährleistet die Vertraulichkeit.

In der Abbildung [6.7] sind die Schritte des RSA-Algorithmus systematisch aufgeführt

❶ Schlüsselerzeugung

① Wähle zwei zufällige Primzahlen p und q, $p \neq q$. Die Primzahlen p und q sind beispielsweise 512 Bits lang.

② Berechne das Produkt $n = p \cdot q$.

③ Berechne die Totientenfunktion $\phi(n) = (p-1) \cdot (q-1)$.

④ Wähle eine positive ganze Zahl e, coprim zu $\phi(n) = (p-1) \cdot (q-1)$.

⑤ Berechne d aus $d \equiv e^{-1} \mod (p-1)(q-1)$

⑥ Der **Public-Key** K_{pub} ist das Zahlenpaar $\mathrm{K}_{pub} = [e, n]$.

⑦ Der **Private-Key** oder **geheime Schlüssel** K_{priv} ist das Zahlenpaar $\mathrm{K}_{priv} = [d, n]$.

❷ Verschlüsselung

① Codiere den Klartext so, dass $M < n$.

② Chiffrieren des Klartexts und Erzeugung des Chiffrats C durch:

$$C = M^e \bmod n$$

❸ Entschlüsselung

① Empfang des Chiffretexts C.

② Dechiffrieren der Nachricht durch Anwendung der Regel

$$M = C^d \bmod n$$

Abbildung 6.7: Der RSA Algorithmus.

Das RSA-Kryptosystem operiert auf der Menge $\mathbb{Z}_n$.

Theorem: (Korrektheit des RSA Algorithmus)

Die RSA Gleichungen

$$P(M) = M^e \bmod n$$
$$S(C) = C^d \bmod n$$

definieren inverse Transformationen auf $\mathbb{Z}_n$ und erfüllen für jede Nachricht $M \in \mathcal{D}$:

$$M = S(P(M))$$
$$M = P(S(M))$$

i.e. der Public-Key P und der Private-Key S spezifizieren Funktionen, die invers zueinander sind.

Beweis:

Für jede Nachricht $M \in \mathbb{Z}_n$ gilt per Konstruktion:

$$P(S(M)) = S(P(M)) = M^{ed} \bmod n$$

Da die Zahlen e und d multiplikativ invers modulo $\phi(n) = (p-1)(q-1)$ sind, haben wir

$$e \cdot d = 1 + k \cdot (p-1) \cdot (q-1)$$

für eine ganze Zahl k. Falls $M \not\equiv 0 \bmod p$ erhalten wir:

$$\begin{aligned} M^{ed} &\equiv M^{1+k(p-1)(q-1)} \bmod p \\ &\equiv M \cdot (M^{p-1})^{k(q-1)} \bmod p \\ &\overset{\text{Fermat}}{\equiv} M \cdot (1)^{k(q-1)} \bmod p \\ &\equiv M \bmod p \end{aligned}$$

wobei das FERMATsche Theorem verwendet wird. Dieses Resultat gilt auch für den Fall, dass $M \equiv 0 \bmod p$. Daher schließen wir:

$$M^{ed} \equiv M \bmod p$$

für alle $M \in \mathcal{D}$. Analog findet man

$$M^{ed} \equiv M \bmod q$$

Nun folgt aufgrund eines Corollars des Chinesischen Restesatzes, falls $M^{ed} \equiv M \bmod p$ und $M^{ed} \equiv M \bmod q$, dann gilt

$$M^{ed} \equiv M \bmod n$$

mit $n = pq$.

Beispiel:

In diesem Beispiel gehen wir die Schritte des RSA Algorithmus mit kleinen Zahlen durch, um zu sehen, wie dieser Algorithmus arbeitet.

① Erzeugung der Schlüssel

(a) Wir wählen zwei Primzahlen, $p = 7$ and $q = 17$.

(b) Wir berechnen das Produkt $n = pq$, i.e. $n = 7 \cdot 17 = 119$.

(c) Die Berechnung der Eulerschen Totientenfunktion ergibt:

$$\begin{aligned}\phi(n) &= \phi(p)\phi(q)\\ &= (p-1)\cdot(q-1)\\ &= \phi(119)\\ &= \phi(7)\cdot\phi(17)\\ &= 6\cdot 16 = 96.\end{aligned}$$

(d) Wie wählen eine Zahl e, so dass e coprim zu $\phi(n) = 96$ und kleiner als $\phi(n)$ ist; wir wählen für dieses Beispiel $e = 5$ mit $\mathtt{ggT}(\phi(n), e) = 1$.

(e) Nun muss d so bestimmt werden, dass

$$d \cdot e \equiv 1 \bmod 96$$

Hier muss der erweiterte Euklidische Algorithmus angewendet werden, mit dessen Hilfe das multiplikative Inverse von e modulo $\phi(n)$ systematisch bestimmt werden kann.

Wir führen die Schritte des Algorithmus explizit aus (siehe Kapitel [18.2] mit $f = 96$ und $d = 5$):

STEP 1 ExE0: Initialisierung

$$\begin{aligned}(X_1, X_2, X_3) &\leftarrow (1, 0, 96)\\ (Y_1, Y_2, Y_3) &\leftarrow (0, 1, 5).\end{aligned}$$

Wir haben also: $X_1 = 1, X_2 = 0, X_3 = 96, Y_1 = 0, Y_2 = 1, Y_3 = 5$.

STEP 2 ExE1: $Y_3 = 0$? nein

STEP 3 ExE2: $Y_3 = 1$? nein

STEP 4 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{96}{5} \rfloor = 19$.

STEP 5 ExE4: Berechne

$$\begin{aligned}T_1 &= X_1 - Q \cdot Y_1 = 1 - 19 \cdot 0 = 1\\ T_2 &= X_2 - Q \cdot Y_2 = 0 - 19 \cdot 1 = -19\\ T_3 &= X_3 - Q \cdot Y_3 = 96 - 19 \cdot 5 = 1\end{aligned}$$

STEP 6 ExE5, **ExE6** und **ExE7**: Ersetze

$$\begin{aligned}X_1 &\leftarrow Y_1 = 0\\ X_2 &\leftarrow Y_2 = 1\\ X_3 &\leftarrow Y_3 = 5\end{aligned}$$

und

$$Y_1 \leftarrow T_1 = 1$$
$$Y_2 \leftarrow T_2 = -19$$
$$Y_3 \leftarrow T_3 = 1$$

Nach diesem Durchgang haben wir also: $X_1 = 0, X_2 = 1, X_3 = 5, Y_1 = 1, Y_2 = -19, Y_3 = 1$. Weiter mit **ExE1**.

STEP 7 ExE1: $Y_3 = 0$? nein

STEP 8 ExE2: $Y_3 = 1$? ja

Damit: $\texttt{ggT}(96, 5) = 1$ und das multiplikative Inverse von 5 mod 96 ist -19.

Der Wert von d, mit dem weiter gearbeitet wird, ist $d = 77$, dieser Wert liegt in der gleichen Restklasse wie -19, da $77 \cdot 5 = 385 = 4 \cdot 96 + 1$.

Damit sind die resultierenden Schlüssel

- **Private Key:** $\mathrm{K}_{priv} = [d, n] = [77, 119]$
- **Public Key:** $\mathrm{K}_{pub} = [e, n] = [5, 119]$

② VERSCHLÜSSELUNG:

Wir wollen nun eine Nachricht mit Hilfe des Public-Keys verschlüsseln. Der Klartext ist der Buchstabe `M`. Gemäß RSA Algorithmus ist dieser Buchstabe als Zahl zu codieren. Dies kann auf vielfältige Art und Weise geschehen, wir verwenden hier den ASCII-Wert, ASCII(`M`) = 77, alternativ kann auch benutzt werden, dass `M` Buchstabe Nummer 13 ist. Wir verwenden in diesem Beispiel der ASCII-Code. Dann ist nach dem RSA Algorithmus die Bedingung $M < n$ erfüllt und für die Verschlüsselung berechnet man die fünfte Potenz von 77. Es folgt:

$$\begin{aligned} 77^5 \bmod 119 &= \big((77^2 \bmod 119) \cdot (77^2 \bmod 119) \cdot (77 \bmod 119)\big) \bmod 119 \\ &= \big((5929 \bmod 119) \cdot (5929 \bmod 119) \cdot \\ &\qquad (77 \bmod 119)\big) \bmod 119 \\ &= \big(98 \cdot 98 \cdot 77\big) \bmod 119 \\ &= 42. \end{aligned}$$

Damit,

$$77^5 \equiv 42 \bmod 119$$

und der Chiffretext ist 42, der zum Empfänger der Nachricht übertragen wird.

③ ENTSCHLÜSSELUNG:

Wir nehmen nun an, wir haben die verschlüsselte Nachricht $C = 42$ erhalten. Wir müssen den geheimen, privaten Schlüssel $\mathrm{K}_{priv} = [d, n] = [77, 119]$ verwenden, um diese Nachricht zu dechiffrieren. Die Aufgabe besteht also darin,

$$M \equiv C^d \bmod n$$

zu berechnen, in unserem Beispiel:

$$M \equiv 42^{77} \bmod 119$$

Wir wenden den `fastexp` Algorithmus an (siehe Kapitel [15.3.3], um das Monster $42^{77} \bmod 119$ zu berechnen.

Zunächst:

$$42^{77} = 42^{64} \cdot 42^8 \cdot 42^4 \cdot 42$$

Daher berechnen wir zunächst die folgenden Potenzen, wir machen dabei Gebrauch von grundlegenden Eigenschaften der modularen Arithmetik.

$$\begin{aligned}
42^2 \bmod 119 &= 98 \\
42^4 \bmod 119 &= 98^2 \bmod 119 = 84 \\
42^8 \bmod 119 &= 84^2 \bmod 119 = 35 \\
42^{16} \bmod 119 &= 35^2 \bmod 119 = 35 \\
42^{32} \bmod 119 &= 35^2 \bmod 119 = 35 \\
42^{64} \bmod 119 &= 35^2 \bmod 119 = 35
\end{aligned}$$

Damit erhalten wir:

$$42^{77} \bmod 119 = (35 \cdot 35 \cdot 84 \cdot 42) \bmod 119 = 77$$

i.e. der ursprüngliche Klartext wird rekonstruiert.

6.4 Aspekte des RSA-Algorithmus

In diesem Abschnitt betrachten wir die Komplexität der Berechnungen, die im RSA Algorithmus eingesetzt werden. Dabei sind zwei Punkte näher zu untersuchen:

- Ver– und Entschlüsselung
- Schlüsselerzeugung

6.4.1 Ver– und Entschlüsselung

Wie wir in dem obigen Beispiel gesehen haben, erfordert sowohl die Ver– als auch die Entschlüsselung des RSA Algorithmus die Potenzierung einer Zahl Modulo n, wie z.B. 42^{77}. Falls man zuerst die Potenz der Zahl berechnet und erst anschließend Modulo n reduziert, entstehen gigantische Zahlen als Zwischenergebnisse. Bei diesem Schritt bietet die modulare Arithmetik ein sehr effektives Verfahren, das im wesentlichen auf der Regel

$$(a \cdot b) \bmod n \equiv \left[(a \bmod n) \cdot (b \bmod n)\right] \bmod n$$

beruht. Daher können beliebige Zwischenergebnisse bereits Modulo n reduziert werden, was die Berechnung überhaupt durchführbar macht.

Ein zweiter Aspekt ist die Effektivität der Potenzierung. Um zu sehen, wie die Effektivität anwächst, betrachten wir die Art und Weise, wie wir 42^{77} berechnen. Eine direkte Berechnung erfordert 77 Multiplikationen:

$$42^{77} = \underbrace{42 \cdot 42 \cdot \ldots \cdot 42}_{77 \text{ times}}$$

Wie wir jedoch in obigem Beispiel gesehen haben, erhalten wir das gleiche Resultat mit weniger als zehn Multiplikationen, falls wiederholt das Quadrat jedes Zwischenergebnisses berechnet wird, man berechnet als sukzessive die Potenzen $42^2, 42^4, 42^8, \ldots$.

Im allgemeinen funktioniert dies folgendermaßen: Angenommen, wir möchten den Wert a^m berechnen mit zwei positiven ganzen Zahlen a und m. Drücken wir m als binäre Zahl mit Koeffizienten $b_k b_{k-1} \ldots b_1 b_0, b_i \in \mathbb{Z}_2$ aus in der Form

$$m = \sum_{b_i \neq 0} 2^i,$$

dann folgt:

$$a^m = a^{\sum_{b_i \neq 0} 2^i} = \prod_{b_i \neq 0} a^{2^i}$$

und

$$a^m \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)}\right] \bmod n = \prod_{b_i \neq 0} \left\{a^{(2^i)} \bmod n\right\}$$

6.4.2 Schlüsselerzeugung

Bevor das Public-Key Kryptosystem angewendet werden kann, muß jede Partei, die das System nutzt ein Schlüsselpaar erzeugen. Diese Prozedur involviert zwei Schritte:

- Bestimmung der beiden Primzahlen p und q.
- Die Wahl von e oder d und Berechnung der anderen Zahl.

Wir betrachten zunächst die Festlegung der beiden Primzahlen p und q etwas näher. Aufgrund der Tatsache, dass $n = p \cdot q$ Teil des Public Keys ist, ist diese Zahl auch einem potentiellen Angreifer bekannt. Daher müssen diese Primzahlen ausreichend groß sein, um die Faktorisierung von n in diese beiden Primzahlen vom Rechenaufwand gesehen unmöglich zu machen. Andererseits muß das Verfahren, ausreichend große Primzahlen zu finden, in vernünftiger Zeit durchführbar sein.

Unglücklicherweise gibt es kein brauchbares Verfahren, das beliebig große Primzahlen generiert, da kein effizienter Algorithmus bekannt ist, um Primzahlen zu erzeugen. Daher ist eine andere Methode notwendig, um geeignet große Primzahlen zu erzeugen. Das übliche Prozedere besteht grob aus den Schritten:

- ✎ wähle eine zufällige, ungerade Zahl der gewünschten Größenordnung
- ✎ teste, ob diese Zahl prim ist oder nicht; falls nicht, wähle eine andere, zufällige Zahl, bis die Tests ergeben, dass die gewählte Zahl (wahrscheinlich) prim ist.

In der Vergangenheit wurden eine große Anzahl verschiedener Verfahren entwickelt, mit denen getestet werden kann, ob eine Zahl prim ist oder nicht. Eine allgemeine Eigenschaft fast aller dieser Tests ist, dass sie auf probabalistischen Verfahren beruhen. Eine große Sammlung solcher Algorithmen findet man in [126] und [172]. Im Prinzip bedeutet dies also, dass diese Tests festlegen, dass eine gegebene Zahl *wahrscheinlich* prim ist. Trotz dieses Mangels an Gewissheit können diese Tests zu durchgeführt werden, dass diese Wahrscheinlichkeit beliebig nahe an 1 kommt.

Eine der Methoden, zu prüfen, ob eine gegebene Zahl prim ist, ist der **Miller-Rabin** Algorithmus (siehe Abschnitt [19.3.3]). Mit diesem Algorithmus – das Prinzip ist bei fast allen Primzahltests ähnlich – wird mit einer gegebenen Zahl n eine Berechnung durchgeführt, die eben diese zu testende Zahl n betrifft sowie eine andere, beliebig gewählte Zahl a. Diese nennt man Zeuge (engl.: *wittness*). Falls n den Test nicht besteht, ist n sicherlich keine Primzahl. Besteht n den Test, kann n Primzahl sein oder nicht. Falls die Zahl n viele dieser Tests mit vielen beliebig und zufällig gewählten Zeugen a besteht, ist die Wahrscheinlichkeit sehr hoch, dass n tatsächlich eine Primzahl ist.

Zusammenfassend besteht das Verfahren, eine Primzahl auszuwählen aus folgenden Schritten:

→ Wähle eine ungerade, zufällige Zahl n mit Hilfe eines geeigneten, kryptographisch sicheren Pseudozufallszahlengenerators.

→ Wähle eine zufällige Zahl $a < n$.

→ Führe einen probabalistischen Primzahltest durch, e.g. MILLER-RABIN. Falls n den Test nicht besteht verwerfe den Wert von n und gehe zum ersten Schritt zurück.

→ Falls n eine geeignete Anzahl von Testläufen besteht, akzeptiere n, andernfalls gehe zu Schritt 2.

Dies scheint eine etwas mühsame Prozedur zu sein. Man mache sich jedoch klar, dass dieser Prozess relativ selten ausgeführt werden muss, denn neue Schlüsselpaare müssen relativ selten generiert werden.

Es ist interessant, die Frage in etwas mehr Detail zu untersuchen, wie viele Zahlen verworfen werden, bevor eine Primzahl gefunden wird. Der Primzahlsatz (siehe [142] oder [195]) sagt aus, dass im Mittel um eine Zahl N (ln (N)) ganze Zahlen verteilt sind. Daher hat man im Durchschnitt die Größenordnung von $\ln(N)$ Zahlen zu testen, bis eine Primzahl gefunden ist. Da alle geraden Zahlen nicht betrachtet werden müssen, ist die genauere Zahl $\ln(N)/2$. Wird beispielsweise eine Primzahl der Größenordnung 2^{200} gesucht, sind im Durchschnitt $\ln(2^{200})/2 = 70$ Versuche notwendig, eine Primzahl zu finden.

Hat man die Primzahlen p und q gefunden, dann wird der Algorithmus zur Schlüsselerzeugung damit beendet, dass ein Wert für e festgelegt und dann der Wert für d berechnet wird (oder umgekehrt). Nimmt man die erste Variante, muß also ein e gewählt werden, so dass $\texttt{ggT}(\phi(n), e) = 1$. Anschließend wird das multiplikative Inverse d berechnet über

$$d = e^{-1} \bmod \phi(n)$$

Der erweiterte EUKLIDische Algorithmus berechnet den größten gemeinsamen Teiler zweier Zahlen, wenn dieser 1 ist, liefert dieser Algorithmus das multiplikative Inverse des einen Faktors zu dem anderen Modulo $\phi(n)$.

6.4.3 Sicherheitsaspekte des RSA Algorithmus

Man kann drei Möglichkeiten untersuchen, den RSA Algorithmus anzugreifen.

- **Brute force Attacken** Die Angriffsart der Brute Force Attacke besteht darin, sämtliche möglichen privaten Schlüssel auszuprobieren.

- **Mathematische Attacken**
  Es gibt eine Vielzahl mathematischer Angriffsarten, sämtliche Attacken beruhen auf der Faktorisierung des Produktes der beiden Primzahlen. Der entscheidende Punkt ist, dass $n = p \cdot q$ einem potentiellen Angreifer als Teil des öffentlichen Schlüssels bekannt ist. Falls es ihm gelingt, aus n die beiden Faktoren p und q zu extrahieren, ist er in der Lage, die Totientenfunktion $\phi(n)$ zu berechnen. Da ihm die Zahl e als zweiter Teil des Public Keys ebenfalls bekannt ist, kann er aus e und $\phi(n)$ mit dem erweiterten

EUKLIDischen Algorithmus die Zahl d bestimmen, damit ist das Kryptosystem gebrochen, da der Angreifer dann im Besitz des privaten Schlüssels ist.

- **Timing Attacken**
  Timing Attacken versuchen, über eine Analyse der Laufzeit des Entschlüsselungsalgorithmus Informationen über der private Key zu erhalten.

Um den RSA Algorithmus gegen Brute Force Angriffe unangreifbar zu machen, verfährt man hier analog zu anderen Kryptosystemen, man wählt einen geeignet großen Schlüssel. Das heißt, je größer die Anzahl der Bits in e und d ist, desto sicherer ist der Algorithmus gegenüber Brute Force Angriffen. Da die Berechnungen zur Schlüsselpaarerzeugung und Ver–/Entschlüsselung sehr komplex sind, hat eine größere Schlüssellänge zur Folge, dass eine längere Laufzeit erforderlich ist.

Um den RSA Algorithmus mit mathematischen Methoden anzugreifen, stehen drei Möglichkeiten zur Verfügung:

- Faktorisierung von n in die beiden Primfaktoren. Dies ermöglicht die berechnung der Totientenfunktion $\phi(n) = (p-1) \cdot (q-1)$, was wiederum die bestimmung von $d = e^{-1} \bmod \phi(n)$ ermöglicht.
- Direkte bestimmung von $\phi(n)$ ohne Faktorisierung von n in p und q. Auch hier erhält ein Angreifer die Möglichkeit, $d = e^{-1} \bmod \phi(n)$ zu finden.
- Direkte Bestimmung von d ohne die Totientenfunktion $\phi(n)$.

Die häufigsten Diskussionen der Kryptoanalyse von RSA beziehen sich auf die Faktorisierung von n in die beiden Primfaktoren p und q. Der Grund liegt darin, dass

1. Die Bestimmung von $\phi(n)$ bei gegebenem n ist äquivalent zur Faktorisierung von n.
2. Mit den zur Zeit bekannten Algorithmen ist die Bestimmung von d bei gegebenem e und n mindestens so zeitaufwändig wie das Faktorisierungsproblem.

Die RSA Challenge (siehe Abschnitt [19.1.1]) ist der Versuch, die Grenzen des Faktorisierungsproblems zu finden.

6.5 Das Diffie-Hellman Key Exchange Verfahren

Der erste veröffentlichte Public-Key Algorithmus erschien in der Arbeit von DIFFIE und HELLMAN [64]. Dieser Algorithmus nennt man üblicherweise **Diffie-Hellman Schlüsselaustausch**. Eine Vielzahl kommerzieller Produkte macht heutzutage Gebrauch dieses Schlüsselaustauschverfahrens.

Zweck des DIFFIE-HELLMAN Algorithmus ist es, dass zwei Benutzer auf sichere Weise – ohne sich direkt zu begegnen oder ohne sich je zuvor begegnet zu sein – einen Schlüssel austauschen können, der dazu benutzt werden kann, mit Hilfe eines symmetrischen Verfahrens Nutzdaten zu verschlüsseln. Der Algorithmus selbst beschränkt sich auf den Austausch des Schlüssels. Der DIFFIE-HELLMAN Algorithmus hat für den praktischen Einsatz einen Nachteil: *Beide* Parteien müssen Informationen austauschen, bevor eine Kommunikation stattfinden kann. Daher ist dieses Verfahren für Telefon-ähnliche Kommunikation geeignet, aber weniger für e-mail Verschlüsselung (da dabei der Empfänger nicht anwesend sein muss) oder eine andere Form der Ein-Weg Kommunikation.

Die Sicherheit des DIFFIE-HELLMAN Algorithmus hängt davon ab, dass es sehr schwierig ist – vom Rechenaufwand betrachtet – diskrete Logarithmen zu berechnen. Diskrete Logarithmen können folgendermaßen definiert werden (Details siehe Kapitel [20]):

Wir definieren zunächst die primitive Wurzel einer Primzahl p als eine Zahl, deren Potenzen (mod p) alle Zahlen von 1 bis $p-1$ generiert. Das bedeutet, falls a eine primitive Wurzel der Primzahl p ist, dann sind die Zahlen

$$a \bmod p; a^2 \bmod p; \cdots ; a^{p-1} \bmod p;$$

unterschiedlich und bestehen aus den ganzen Zahlen von 1 bis $p-1$ in irgend einer Reihenfolge.

Für jede ganze Zahl b und eine primitive Wurzel a einer Primzahl p kann man einen eindeutigen Exponenten i finden, so dass

$$b = a^i \bmod p \qquad 0 \leq i \leq p-1$$

Der Exponent i ist der **diskrete Logarithmus** – oder **Index** – von b für die Basis a mod p. Diesen Wert bezeichnet man mit $\text{ind}_{a,p}(b)$.

Mit dieser Hintergrundinformation können wir den DIFFIE-HELLMAN Algorithmus definieren. Die grundlegenden Schritte sind die folgenden:

Diffie-Hellman Key Exchange Kryptosystem

❶ Szenario:
Alice und Bob wollen einen gemeinsamen Schlüssel erzeugen, den nur sie kennen ohne sich zuvor begegnet zu sein.

❷ Globale öffentliche Parameter
Alice und Bob einigen sich öffentlich – i.e. diese Kommunikation kann über einen unsicheren Kanal erfolgen – auf einen Wert für eine Primzahl q und einen Wert a, $a < q$, a ist primitive Wurzel von q. Sowohl Alice als auch Bob kennen also die Werte von q und a.

❸ Geheimer Schlüssel Alice
Alice wählt eine Zahl X_A mit $X_A < q$; der Wert von X_A ist nur Alice bekannt. Sie berechnet

$$Y_A = a^{X_A} \bmod q$$

Alice sendet den Wert Y_A – dieser ist also öffentlich – an Bob.

❹ Geheimer Schlüssel Bob
Bob wählt eine Zahl X_iB mit $X_B < q$; der Wert von X_B ist nur Bob bekannt. Er berechnet

$$Y_B = a^{X_B} \bmod q$$

Bob sendet den Wert Y_B – dieser ist ebenfalls öffentlich – an Alice.

❺ Geheime Schlüsselerzeugung durch Alice
Alice berechnet mit ihrem geheimen Schlüssel X_A und dem öffentlichen, von Bob stammenden Wert Y_B den Wert

$$K = (Y_B)^{X_A} \bmod q$$

❻ Geheime Schlüsselerzeugung durch Bob
Bob berechnet mit seinem geheimen Schlüssel X_B und dem öffentlichen, von Alice stammenden Wert Y_A den Wert

$$K' = (Y_A)^{X_B} \bmod q$$

Für die von Alice und Bob berechneten Werte K und K' gilt jedoch

$$K = K',$$

Mit Hilfe der modularen Arithmetik zeigt man leicht:

$$\begin{aligned} K &= Y_B^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\ &= (\alpha^{X_B})^{X_A} \bmod q \\ &= \alpha^{X_B \cdot X_A} \bmod q \\ &= (\alpha^{X_A})^{X_B} \bmod q \\ &= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\ &= Y_A^{X_B} \bmod q \\ &= K \end{aligned}$$

Mit anderen Worten, die beiden Kommunikationspartner verfügen nun über einen Schlüssel K, den nur sie kennen. Dieser Schlüssel K kann nun zur Verschlüsselung von Nachrichten durch ein beliebiges symmetrisches Verfahren genutzt werden. Alice und Bob teilen daher ein gemeinsames Geheimnis, ohne sich vorher begegnet zu sein. Da die beiden Werte X_A und X_B geheim sind, hat ein potentieller Angreifer die folgenden Werte, um das System zu kompromittieren:

- die öffentlich zugängliche Primzahl q
- die öffentlich zugängliche primitive Wurzel α
- die öffentlich bekannte Zahl Y_A
- und die öffentlich bekannte Zahl Y_B

Daher ist der Angreifer gezwungen, den diskreten Logarithmus zu bilden, um den geheimen Schlüssel X_A oder X_B eines Kommunikationspartners zu erhalten. Falls der Angreifer beabsichtigt, Bobs Schlüssel zu brechen, muss er folgendes berechnen:

$$X_B = \operatorname{ind}_{\alpha,q}(Y_B)$$

Der Angreifer kann aus diesem Wert den Schlüssel K dann genauso berechnen wie Bob.

Anmerkungen:

① Die Sicherheit des DIFFIE-HELLMAN Key Exchange Verfahrens basiert auf der Tatsache, dass es relativ leicht ist – i.e. es sind effektive Algorithmen bekann wie z.B. der `fastexp` Algorithmus (siehe Abschnitt [15.3.3]) – die Exponentiation modulo einer Primzahl auszuführen. Andererseits wird das Umkehrproblem – i.e. das hier vorliegende **diskrete Logarithmusproblem** – als hartes Problem angesehen. Es ist also kein effektiver Algorithmus bekannt[5], mit dessen Hilfe der diskrete Logarithmus in polynomialer Laufzeit berechnet werden kann. Für große Primzahlen wird das diskrete Logarithmusproblem als undurchführbar angesehen.

[5] Es ist auch nicht bekannt oder mathematisch bewiesen, ob es solch einen Algorithmus überhaupt gibt oder nicht.

② Wenn Alice und Bob das so beschriebene DIFFIE-HELLMAN Verfahren anwenden, verfügen sie anschließend über einen Session Key, mit dem Nachrichten durch symmetrische Chiffrierverfahren verschlüsselt werden können. Dies bietet in dieser Form keine Gewährleistung einer sicheren Kommunikation, daher kann das DIFFIE-HELLMAN Verfahren in dieser Form in der Praxis nicht eingesetzt werden [67, pp. 36–38]. Denn weder Bob noch Alice verfügen über eine sichere Kenntnis der wirklichen Identität des Kommunikationspartners. Sie können sich noch nicht einmal sicher sein, ob nicht ein Angreifer eine sogenannte *man-in-the-middle* Attacke ausführt, in dem dieser mit Alice und Bob jeweils einen Key-Exchange ausgeführt hat und die Kommunikation zwischen Alice und Bob in Klartextform aufzeichnet.

Der notwendige zweite Schritt, um mit Hilfe des DIFFIE-HELLMAN Verfahrens eine sichere Kommunikation zwischen zwei Parteien aufzubauen, ist der Austausch von sogenannten **Zertifikaten**[6]. Alice sendet ihr Zertifikat an Bob und umgekehrt sendet Bob sein Zertifikat an Alice. Dieses Dokument beweist nun auch noch nicht die Echtheit des Absenders. Daher überprüfen die Empfänger der Zertifikate die Authentizität dieser Dokumente anhand der Signatur der ausstellenden Zertifikatstelle. Dadurch weiß Alice, daß das erhaltene Zertifikat echt ist und von der angegebenen Zertifikatstelle stammt und Bob seinerseits weiß, daß das Zertifikat, welches er hat ebenfalls echt ist und ebenfalls von der angegebenen Zertifikatstelle stammt. Was beide noch nicht wissen ist die Authentizität der Person, die das Zertifikat gesendet hat.

In einem letzten Schritt muß (beispielsweise) Alice verifizieren, daß die Person, mit der sie in Kontakt ist, der legitime Eigentümer von Bobs Zertifikat ist. Sie muß daher verfizieren, daß die Person, mit der sie kommuniziert, den private Key besitzt, der zu dem Public-Key des Zertifikats passt. Dieser letzte Schritt wird durch eine Verfahren mit der Bezeichnung **Challenge and Response** ausgeführt. Bei Challenge and Response Verfahren sendet die eine Partei ein Testnachricht und beurteilt die Legitimität der anderen Partei anhand der Verifikation der Signatur der Antwort.

③ Der oben vorgestellte DIFFIE-HELLMAN Algorithmus hat einen weiteren Nachteil. *Beide* Parteien müssen Informationen austauschen, bevor eine Kommunikation stattfinden kann. Daher ist das Verfahren geeignet für Kommunikationsformen, in denen beide Parteien 'Online' sind wie beispielsweise bei einem Telefonat. Weniger geeignet ist das Verfahren bei Einweg-Kommunikationen wie E-Mails.

Beispiel:

[6] Ein Zertifikat ist eine signierte und mit Zeitstempel versehene Message, die von einer (zentralen) Schlüssel-Management Einheit – eine wesentliche Komponente einer *Public-Key Infrastruktur* (PKI) an den Teilnehmer gesendet wird. Das Zertifikat enthält unter anderem der Public Key des Teilnehmers sowie identifizierende Informationen wie Name und Adresse. Wenn zwei Teilnehmer der PKI eine Kommunikaton aufbauen, werden diese Zertifikate zunächst ausgetauscht. Jeder Teilnehmer verifiziert die Signatur der Schlüssel-Management Einheit auf dem erhaltenen Zertifikat und überzeugt sich so über die Authentizität des Dokumentes.

Alice und Bob einigen sich auf folgende öffentliche Parameter

$$q = 479, \qquad a = 11$$

Alice wählt die Zahl: $X_A = 102$

Bob wählt die Zahl: $X_B = 110$

Alice berechnet die Zahl

$$\begin{aligned} Y_A &= a^{X_A} \bmod q \\ &= 11^{102} \bmod 479 \end{aligned}$$

mit Hilfe des `fastexp` Algorithmus folgt:

$$\begin{aligned} 11^2 &\equiv 121 \bmod 479 \\ 11^4 &\equiv 121^2 \bmod 479 \equiv 271 \bmod 479 \\ 11^8 &\equiv 271^2 \bmod 479 \equiv 154 \bmod 479 \\ 11^{16} &\equiv 154^2 \bmod 479 \equiv 245 \bmod 479 \\ 11^{32} &\equiv 245^2 \bmod 479 \equiv 150 \bmod 479 \\ 11^{64} &\equiv 150^2 \bmod 479 \equiv 466 \bmod 479 \end{aligned}$$

Da

$$\begin{aligned} 11^{102} \bmod 479 &= 11^{64+32+4+2} \bmod 479 \\ &= \left(11^{64} \times 11^{32} \times 11^4 \times 11^2\right) \bmod 479 \\ &= \left(466 \times 150 \times 271 \times 121\right) \bmod 479 \\ &= 218 \bmod 479. \end{aligned}$$

überträgt Alice die Zahl

$$Y_A = 218$$

an Bob.

Bob seinerseits berechnet nun die Zahl

$$\begin{aligned} Y_B &= a^{X_B} \bmod q \\ &= 11^{110} \bmod 479 \end{aligned}$$

Da

$$\begin{aligned} 11^{110} \bmod 479 &= 11^{64+32+8+4+2} \bmod 479 \\ &= \left(11^{64} \times 11^{32} \times 11^8 \times 11^4 \times 11^2\right) \bmod 479 \\ &= \left(466 \times 150 \times 154 \times 271 \times 121\right) \bmod 479 \\ &= 218 \times 154 \bmod 479 \\ &= 42 \bmod 479 \end{aligned}$$

überträgt Bob die Zahl

$$Y_B = 42$$

an Alice.

Alice berechnet nun den Session Key gemäß:

$$\begin{aligned} K &= Y_B^{X_A} \bmod 479 \\ &= 42^{102} \bmod 479 \end{aligned}$$

Mit

$$\begin{aligned} 42^2 &\equiv 327 \bmod 479 \\ 42^4 &\equiv (42^2)^2 \bmod 479 \equiv 327^2 \bmod 479 \equiv 112 \bmod 479 \\ 42^8 &\equiv (42^4)^2 \bmod 479 \equiv 112^2 \bmod 479 \equiv 90 \bmod 479 \\ 42^{16} &\equiv (42^8)^2 \bmod 479 \equiv 90^2 \bmod 479 \equiv 436 \bmod 479 \\ 42^{32} &\equiv (42^{16})^2 \bmod 479 \equiv 436^2 \bmod 479 \equiv 412 \bmod 479 \\ 42^{64} &\equiv (42^{32})^2 \bmod 479 \equiv 412^2 \bmod 479 \equiv 178 \bmod 479 \end{aligned}$$

folgt

$$\begin{aligned} 42^{102} \bmod 479 &= 42^{64+32+4+2} \bmod 479 \\ &= \left(42^{64} \times 42^{32} \times 42^4 \times 42^2\right) \bmod 479 \\ &= \left(178 \times 412 \times 112 \times 327\right) \bmod 479 \\ &= 242 \bmod 479. \end{aligned}$$

Bob berechnet:

$$\begin{aligned} K' &= Y_A^{X_B} \bmod 479 \\ &= 218^{110} \bmod 479 \end{aligned}$$

Mit

$$\begin{aligned} 218^2 &\equiv 103 \bmod 479 \\ 218^4 &\equiv (218^2)^2 \bmod 479 \equiv 103^2 \bmod 479 \equiv 71 \bmod 479 \\ 218^8 &\equiv (218^4)^2 \bmod 479 \equiv 71^2 \bmod 479 \equiv 251 \bmod 479 \\ 218^{16} &\equiv (218^8)^2 \bmod 479 \equiv 251^2 \bmod 479 \equiv 252 \bmod 479 \\ 218^{32} &\equiv (218^{16})^2 \bmod 479 \equiv 252^2 \bmod 479 \equiv 276 \bmod 479 \\ 218^{64} &\equiv (218^{32})^2 \bmod 479 \equiv 276^2 \bmod 479 \equiv 15 \bmod 479 \end{aligned}$$

folgt

$$\begin{aligned} K' &= 218^{110} \bmod 479 \\ &= 218^{64+32+8+4+2} \bmod 479 \\ &= \left(218^{64} \times 218^{32} \times 218^8 \times 218^4 \times 218^2\right) \bmod 479 \\ &= \left(15 \times 276 \times 251 \times 71 \times 103\right) \bmod 479 \\ &= 242 \bmod 479 \\ &= K \end{aligned}$$

Damit ist der Session Key: $K = 242$.

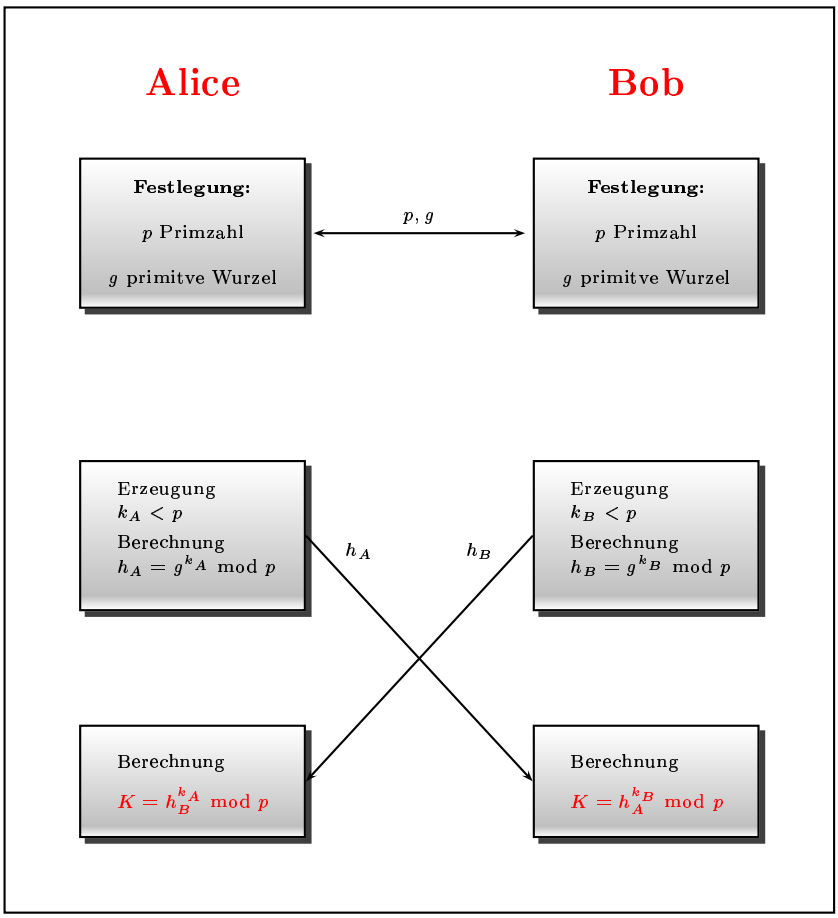


Abbildung 6.8: Protokoll, das den DIFFIE-HELLMAN Key Exchange verwendet.

6.6 ElGamal Kryptosysteme

Die ELGAMAL Verschlüsselungssysteme sind in [75] und [76] beschrieben, [22, 193] ist eine weitere Quelle. Eine umfassende Diskussion verschiedener Aspekte der ELGAMAL Verschlüsselungssysteme findet man in [154, Abschnitt 11.5.2].

Das ELGAMAL Verschlüsselungsverfahren kann sowohl für digitale Signatur als auch für die reine Verschlüsselung eingesetzt werden. Grundlegend hängt die Sicherheit dieses Schemas an der Schwierigkeit, diskrete Logarithmen in endlichen Körpern zu berechnen.

Das ELGAMAL Verfahren wurde von TAHER ELGAMAL (1955 –) im Jahre 1985 in einer Arbeit mit dem Titel

> *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm*

vorgestellt.

Um in dem ElGamal Kryptosystem ein Schlüsselpaar zu erzeugen, wählt man zunächst eine Primzahl p und zwei Zufallszahlen g, x mit $g, x < p$. Anschließend berechnet man

$$y = g^x \bmod p$$

Der öffentliche Schlüssel ist das Paar y, g sowie die Primzahl p. Sowohl g als auch p kann unter einer Gruppe von Benutzern verteilt werden. Der geheime private Schlüssel ist der Exponent x.

6.6.1 Digitale Signatur mit dem ElGamal Verfahren

Das ElGamal Signatur Verfahren stellt ein randomisiertes Signaturverfahren dar. Es generiert digitale Signaturen als als Anhang an binäre Nachrichten beliebiger Länge. In mathematischer Terminologie stellt das ElGamal Verfahren eine Hash Funktion der Form

$$h : \{0,1\}^* \longrightarrow \mathbb{Z}_p$$

dar, wobei p eine große Primzahl ist.

Um eine Nachricht M mit dem ElGamal Verfahren zu unterzeichnen, wählt man zunächst eine zufällige Zahl k, so daß k und $p-1$ coprim sind, i.e.

$$\mathrm{ggT}(k, p-1) = 1$$

. Dann berechnet man:

$$a = g^k \bmod p$$

Anschließend benutzt man den erweiterten Euklidischen Algorithmus um die Gleichung

$$M = (xa + kb) \bmod (p-1)$$

nach b aufzulösen. Die Signatur ist das Paar a und b. Die zufällige Zahl k muß geheim bleiben.

Der Empfänger eine digital signierten Nachricht – Bob – will diese Signatur verifizieren. Dazu muß Bob zeigen, dass gilt:

$$y^a a^b \bmod p = g^M \bmod p$$

Jede einzelne ElGamal Signatur oder Verschlüsselung erfordert die Wahl eines neuen Wertes von k. Dieser Wert wird zufällig bestimmt. Falls ein Angreifer Eve jemals den Wert von k erhält, den Alice verwendet, kann Eve den privaten Schlüssel x von Alice rekonstruieren. Falls Eve jemals zwei Nachrichten erhält, die mit dem gleichen k signiert oder verschlüsselt sind, kann Eve den Wert x rekonstruieren.

Der ElGamal Algorithmus zur digitalen Signatur ist in der Tabelle [6.4] zusammengefaßt.

❶ Public Key

p Primzahl, die an eine Gruppe von Benutzer verteilt werden kann

g zufälliger Wert, der an eine Gruppe von Benutzern verteilt werden kann

y $= g^x \bmod p$

Damit ist der Public Key das Zahlentripel

$$\mathrm{K}_{pub} = [p, g, y]$$

❷ Private Key

x $< p$ Damit ist der Private Key die Zahl

$$\mathrm{K}_{priv} = [x].$$

❸ Erzeugung der Signatur

k zufällig gewählte Zahl, coprim zu $p-1$

a (Signatur) $= g^k \bmod p$

b (Signatur) so dass $M = (xa + kb) \bmod (p-1)$

❹ Verifikation

Als gültige Signatur akzeptiert, falls $y^a a^b \bmod p = g^M \bmod p$

Tabelle 6.4: Digitale Signatur mit dem ElGamal Verfahren.

Beispiel:

Wir betrachten ein Beispiel mit kleinen Zahlen, um ein Gefühl für die Arbeitsweise des ElGamal Signaturverfahrens zu erhalten.

① *Szenario:* Alice will Bob eine digital signierte Nachricht zukommen lassen. Sie verwendet dazu das ElGamal Verfahren.

② Alice wählt

(a) eine Primzahl $p = 13$

(b) eine zufällige Zahl $g < p$, hier $g = 7$.

Diese beiden Zahlen bilden einen Teil des Public Keys.

③ Als Teil des Private Keys wählt Alice eine weitere Zahl $x < p$, hier $x = 3$.

④ Im nächsten Schritt berechnet Alice den Wert

$$y = g^x \bmod p = 7^3 \bmod 13 = 343 \bmod 13 = 5$$

Damit ist der Public Key von Alice das Zahlentripel:

$$\mathrm{K}_{pub}^{A} = [p, g, y] = [13, 7, 5].$$

und der Private Key die Zahl

$$\mathrm{K}_{priv}^{A} = [x] = 3.$$

⑤ Alice möchte nun die Nachricht $M = 10$ signieren. Gemäß der Tabelle [6.4] hat sie dazu die beiden Zahlen a und b zu berechnen.

⑥ Zunächst muß Alice zu diesem Zweck eine zufällige Zahl k wählen, die coprim zu $p - 1 = 12$ ist. Da

$$\mathrm{ggT}(12, 5) = 1$$

ist $k = 5$ eine geeignete Wahl.

⑦ Anschließend berechnet Alice den ersten Wert a der Signatur mit:

$$\begin{aligned} a &= g^k \bmod p \\ &= 7^5 \bmod 13 \\ &= 16\,807 \bmod 13 \\ &\equiv 11 \bmod 13 \end{aligned}$$

⑧ Sind diese Werte verfügbar, kann Alice folgende Gleichung aufstellen:

$$\begin{aligned} M &= (xa + kb) \bmod (p-1) \\ 10 &= (3 \cdot 11 + 5 \cdot b) \bmod 12 \\ 10 &= (33 + 5 \cdot b) \bmod 12 \\ 10 &= 9 + 5 \cdot b \bmod 12 \\ 1 &= 5 \cdot b \bmod 12 \end{aligned}$$

oder

$$b \equiv 5 \bmod 12$$

Damit hat Alice die Signatur generiert, i.e. das Zahlenpaar [11,5].

⑨ Alice sendet nun die Nachricht $M = 10$ und die Signaturwerte $a = 11, b = 5$ an Bob, der verifizieren möchte, dass Alice tatsächlich die Nachricht gesendet hat. Um dieses Ziel zu erreichen, stehen Bob folgende Informationen zur Verfügung:

- → die empfangene Nachricht $M = 10$, diese ist unverschlüsselt
- → die beiden Signaturwerte $[a = 11, b = 5]$
- → den Public Key von Alice $\mathrm{K}_{pub}^{A} = [p = 13, g = 7, y = 5]$

- Mit diesen Werten muß Bob die Gültigkeit der folgenden Gleichung verifizieren:

$$y^a \cdot a^b \bmod p \stackrel{!}{=} g^M \bmod p$$

Die Berechnung der linken Seite liefert das Ergebnis:

$$y^a \cdot a^b \bmod p = 5^{11} \cdot 11^5 \bmod 13 = 4 \bmod 13$$

Die Berechnung der rechten Seite ergibt:

$$g^M \bmod p = 7^{10} \bmod 13 = 4 \bmod 13$$

Da beide Ausdrücke den gleiche Wert ergeben, kann Bob sicher sein, dass die Nachricht M von Alice stammen muß.

6.6.2 Verschlüsselung mit dem ElGamal Kryptosystem

Eine Modifikation des ElGamal Signatur Schemas kann eingesetzt werden, um Nachrichten zu verschlüsseln. Im Gegensatz zu dem RSA Kryptosystem, bei dem sowohl Verschlüsselung als auch Signatur mit dem gleichen Algorithmus – i.e. modulare Exponentiation – ausgeführt wird, benötigt man beim ElGamal Verfahren unterschiedliche Algorithmen zur Durchführung dieser Aufgaben. Die Sicherheit des ElGamal Kryptosystems beruht jedoch in beiden Fällen auf dem diskreten Logarithmusproblem.

ElGamal Public-Key Kryptosystem über $\mathbb{Z}_p^*$

Sei p eine Primzahl, so daß das diskrete Logarithmusproblem in $(\mathbb{Z}_p^*, \cdot)$ undurchführbar ist und sei $g \in \mathbb{Z}_p^*$ eine primitive Wurzel. Klartextraum $\mathcal{P}$ und Chiffretextraum $\mathcal{C}$ seien:

$$\mathcal{P} = \mathbb{Z}_p^*$$
$$\mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$$

Der Schlüsselraum $\mathcal{K}$ ist:

$$\mathcal{K} = \{([p, g, y], [x]) \mid y \equiv g^x \bmod p\}$$

Der Schlüssel ist also eine Vier-Tupel

$$K \in \mathcal{K} \text{ mit } K = (\mathrm{K}_{pub}, \mathrm{K}_{priv}) = ([p, g, y], [x])$$

mit dem Public Key:

$$\mathrm{K}_{pub} = [p, g, y]$$

und dem Private Key:

$$\mathrm{K}_{priv} = [x]$$

Für $K = ([p, g, y], [x])$ und einer geheimen, zufälligen Zahl $k \in \mathbb{Z}_{p-1}$ definieren wir die Verschlüsselung des Klartextes $a \in \mathbb{Z}_p^*$:

$$e_K(a, k) = (y_1, y_2)$$

mit

$$y_1 \equiv g^k \bmod p$$
$$y_2 \equiv a \cdot y^k \bmod p$$

Für $y_1, y_2 \in \mathbb{Z}_p^*$ definieren wir die Dechiffrierfunktion:

$$d_K(y_1, y_2) = y_2 (y_1^x)^{-1} \bmod p$$

Die Verschlüsselungsoperation des ElGamal Kryptosystems ist randomisiert, da der Chiffretext sowohl vom Klartext a als auch von dem zufällig gewählten Wert k abhängt. Daher gibt es viele Chiffretexte (das sind genau $p - 1$), die Verschlüsselungen des gleichen Klartextes sind.

Anmerkung:

Jede ElGamal Verschlüsselung oder Signatur erfordert die Wahl eines neuen k Wertes. Dieser Wert muß zufällig gewählt werden. Falls ein Angreifer zwei Nachrichten abfängt, die mit dem gleichen k Wert verschlüsselt oder signiert wurden, ist der Angreifer in der Lage den Private Key $[x]$ abzuleiten. Dann hat y_1 den gleichen Wert in beiden Chiffraten

Check:

Angenommen, Alice chiffriert zwei Nachrichten M_1 und M_2 um sie an Bob zu senden und benutzt den gleichen Wert k für beide Nachrichten.

Im folgenden gehen wir das ElGamal Verschlüsselungsverfahren Schritt für Schritt explizit noch einmal durch.

Schlüsselerzeugung für ElGamal Kryptosysteme

❶ *Szenario:* Bob will an Alice eine Nachricht M senden, die mit dem ElGamal Kryptosystem chiffriert ist.

❷ Alice generiert dazu ihren Public Key K_{pub}^{A} und den dazu passenden Private Key K_{priv}^{A}.

❸ Alice wählt dazu:

(a) eine große Primzahl p

(b) eine primitive Wurzel g der multiplikativen Gruppe $\mathbb{Z}_p^*$

❹ Alice wählt weiterhin eine zufällige Zahl $x \in \{1, 2, \ldots, p-2\}$ und berechnet den Wert

$$y = g^x \bmod p$$

❺ Damit hat Alice das Schlüsselpaar $(\mathrm{K}_{pub}^{A}, \mathrm{K}_{priv}^{A}) \in \mathcal{K}$ generiert:

$$\mathrm{K}_{pub}^{A} = [p, g, y]$$
$$\mathrm{K}_{priv}^{A} = [x]$$

Verschlüsselung mit ElGamal Kryptosystemen

❶ Bob besorgt sich den authentischen Public Key von Alice

$$\mathrm{K}_{pub}^{A} = [p, g, y].$$

❷ Bob codiert die Nachricht $a \in \mathcal{P}$ als Zahl in der Menge $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$.

❸ Bob wählt eine zufällige Zahl

$$k \in \{1, 2, \ldots, p-2\}$$

❹ Bob berechnet mit den durch den Public Key zur Verfügung stehenden Werten p, g und y die Werte

$$y_1 \equiv g^k \bmod p$$
$$y_2 \equiv a \cdot y^k \bmod p$$

❺ Bob überträgt den Chiffretext

$$C = e_{\mathrm{K}^A_{pub}}(a) = (y_1, y_2) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$$

an Alice.

Entschlüsselung mit ElGamal Kryptosystemen

❶ Alice erhält eine Nachricht $C = (y_1, y_2)$, die mit ihrem Public Key $\mathrm{K}^A_{pub} = [p, g, y]$ chiffriert wurde.

❷ Alice berechnet mit Hilfe ihres Private Keys $\mathrm{K}^A_{priv} = [x]$ den Wert

$$z = (y_1^x)^{-1} \bmod p = \left((g^k)^x\right)^{-1} \bmod p$$

❸ Alice rekonstruiert den Klartext a über

$$a = d_{\mathrm{K}^A_{priv}}(M) = z \cdot y_2 \bmod p$$

Das ElGamal Kryptosystem operiert also auf folgenden Mengen:

- dem Klartextraum $\mathcal{P} = \mathbb{Z}_p$
- dem Chiffretextraum $\mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
- dem Schlüsselraum

$$\{K\} = \{([p, g, y], [x]) \mid p \text{ Primzahl }, g \in \mathbb{Z}_p^*, y \in \mathbb{Z}_p, x \in \mathbb{Z}_{p-1}\}$$

Theorem:

Für jede mit obigem ElGamal Verfahren chiffrierte Nachricht $M \in \mathbb{Z}_p$ gilt:

$$M = d_{\mathrm{K}_{priv}}\left(e_{\mathrm{K}_{pub}}\right)(M)$$

Beweis:

Es gilt:

$$\begin{aligned}
d_{\mathrm{K}_{priv}}\left(e_{\mathrm{K}_{pub}}\right)(M) &= d_{\mathrm{K}_{priv}}(y_1, y_2) \\
&= z \cdot b \bmod p \\
&= a^{p-1-x} \cdot b \bmod p \\
&= (g^k)^{p-1-x} \cdot M \cdot y^k \bmod p \\
&= (g^k)^{p-1-x} \cdot M \cdot (g^x)^k \bmod p \\
&= (g^k)^{p-1-x} \cdot M \cdot (g^k)^x \bmod p \\
&= (g^k)^{p-1} \cdot M \bmod p \\
&= M
\end{aligned}$$

wobei

$$(g^k)^{p-1} \equiv 1 \bmod p$$

nach dem kleinen Satz von FERMAT gilt.

Beispiel:

In diesem Beispiel verwenden wir das ELGAMAL Kryptosystem, um eine Nachricht zu ver– bzw. entschlüsseln.

① Alice erzeugt ein Schlüsselpaar. Dazu wählt sie

$$p = 29$$
$$g = 2$$

g ist eine primitive Wurzel der multiplikativen Gruppe $\mathbb{Z}_{29}^*$, denn jedes Element der Menge $\mathbb{Z}_{29}^*$ wird durch eine Potenz von 2 modulo 29 erzeugt.

Die Elemente der Gruppe $\mathbb{Z}_{29}^*$ sind:

$$\mathbb{Z}_{29}^* = \{1, 2, 3, \ldots, 27, 28\}$$

Check:

$$
\begin{aligned}
2^1 &\bmod 29 \equiv 2 \bmod 29\\
2^2 &\bmod 29 \equiv 4 \bmod 29\\
2^3 &\bmod 29 \equiv 8 \bmod 29\\
2^4 &\bmod 29 \equiv 16 \bmod 29\\
2^5 &\bmod 29 \equiv 3 \bmod 29\\
2^6 &\bmod 29 \equiv 6 \bmod 29\\
2^7 &\bmod 29 \equiv 12 \bmod 29\\
2^8 &\bmod 29 \equiv 24 \bmod 29\\
2^9 &\bmod 29 \equiv 19 \bmod 29\\
2^{10} &\bmod 29 \equiv 9 \bmod 29\\
2^{11} &\bmod 29 \equiv 18 \bmod 29\\
2^{12} &\bmod 29 \equiv 7 \bmod 29\\
2^{13} &\bmod 29 \equiv 14 \bmod 29\\
2^{14} &\bmod 29 \equiv 28 \bmod 29\\
2^{15} &\bmod 29 \equiv 27 \bmod 29\\
2^{16} &\bmod 29 \equiv 25 \bmod 29\\
2^{17} &\bmod 29 \equiv 21 \bmod 29\\
2^{18} &\bmod 29 \equiv 13 \bmod 29\\
2^{19} &\bmod 29 \equiv 26 \bmod 29\\
2^{20} &\bmod 29 \equiv 23 \bmod 29\\
2^{21} &\bmod 29 \equiv 17 \bmod 29\\
2^{22} &\bmod 29 \equiv 5 \bmod 29\\
2^{23} &\bmod 29 \equiv 10 \bmod 29\\
2^{24} &\bmod 29 \equiv 20 \bmod 29\\
2^{25} &\bmod 29 \equiv 11 \bmod 29\\
2^{26} &\bmod 29 \equiv 22 \bmod 29\\
2^{27} &\bmod 29 \equiv 15 \bmod 29\\
2^{28} &\bmod 29 \equiv 1 \bmod 29
\end{aligned}
$$

Dies zeigt explizit, dass die Potenzen des Elements $g = 2 \in \mathbb{Z}_{29}^*$ die Gruppe $\mathbb{Z}_{29}^*$ generiert.

② Alice wählt eine zufällige Zahl $x \in \mathbb{Z}_{p-1}$. Im konkreten Beispiel ist dies die Menge:

$$
\begin{aligned}
\mathbb{Z}_{p-1} &= \mathbb{Z}_{28}\\
&= \{0, 1, 2, \ldots, 25, 26, 27\}
\end{aligned}
$$

Alice wählt den Wert $x = 12$ und berechnet

$$\begin{aligned} y &= g^x \bmod p \\ &= 2^{12} \bmod 29 \\ &= 7 \end{aligned}$$

③ Damit hat Alice das folgende Schlüsselpaar generiert:

$$\begin{aligned} \mathrm{K}_{pub}^{A} &= [p, g, y] = [29, 2, 7] \\ \mathrm{K}_{priv}^{A} &= [x] = [12] \end{aligned}$$

④ Bob besorgt sich den authentischen Public Key von Alice

$$\mathrm{K}_{pub}^{A} = [p, g, y] = [29, 2, 7]$$

⑤ Bob codiert die Nachricht – beispielsweise den Klartextbuchstaben `m` – als Zahl in der Menge $\mathbb{Z}_p = \{0, 1, \ldots, 28\}$. Da `m` der 13. Buchstabe ist, wird diese Zahl als Klartext benutzt.

⑥ Bob wählt eine zufällige Zahl k aus der Menge $\{1, 2, \ldots, 27\}$, zum Beispiel die Zahl $k = 11$.

⑦ Bob berechnet

$$\begin{aligned} y_1 &= g^k \bmod p \\ &= 2^{11} \bmod 29 \\ &= 18 \end{aligned}$$

$$\begin{aligned} y_2 &= M \cdot y^k \bmod p \\ &= (13 \cdot 7^{11}) \bmod 29 \\ &= 9 \end{aligned}$$

⑧ Bob überträgt den Chiffretext

$$C = e_{\mathrm{K}_{pub}^{A}}(M) = (y_1, y_2) = (18, 9)$$

an Alice.

⑨ Alice erhält die Nachricht $C = (y_1, y_2) = (18, 9)$, die mit ihrem Public Key K_{pub}^{A} chiffriert wurde.

⑩ Alice berechnet mit Hilfe ihres Private Keys $\mathrm{K}_{priv}^{A} = [x] = [12]$ den Wert

$$\begin{aligned} z &= (y_1^x)^{-1} \bmod p \\ &= (y_1)^{p-1-x} \bmod p \\ &\equiv 18^{29-1-12} \bmod 29 \\ &\equiv 18^{16} \bmod 29 \\ &\equiv 24 \bmod 29 \end{aligned}$$

❶ Alice rekonstruiert den Klartext über

$$\begin{aligned} M &= d_{\mathrm{K}_{priv}^{A}}(M) \\ &= (z \cdot y_2) \bmod p \\ &= 24 \cdot 9 \bmod 29 \\ &= 13 \end{aligned}$$

Beispiel:

In diesem zweiten Beispiel benutzen wir etwas größere Zahlen und verschlüsseln damit einen etwas längeren Klartext.

① Alice erzeugt ein Schlüsselpaar. Dazu wählt sie zunächst eine Primzahl p und eine primitive Wurzel g:

$$\begin{aligned} p &= 2579 \\ g &= 2 \end{aligned}$$

$g = 2$ ist eine primitive Wurzel modulo 2579.

② In einem zweiten Schritt wählt Alice eine Zahl $x \in \mathbb{Z}_{p-1}$, beispielsweise:

$$x = 765.$$

③ Alice berechnet nun den Wert:

$$\begin{aligned} y &\equiv g^x \bmod p \\ &\equiv 2^{765} \bmod 2579 \\ &\equiv 949 \bmod 2579 \end{aligned}$$

Check:

Wir wenden den `fastexp` Algorithmus (siehe Abschnitt [15.3.3]) an, um y zu berechnen. Der Exponent 765 wird als Summe von 2er Potenzen geschrieben:

$$765 = 512 + 128 + 64 + 32 + 16 + 8 + 4 + 1$$

Nun ist:

$$\begin{aligned} 2^2 \bmod 2579 &\equiv 4 \bmod 2579 \\ 2^4 \bmod 2579 &\equiv 16 \bmod 2579 \\ 2^8 \bmod 2579 &\equiv 256 \bmod 2579 \\ 2^{16} \bmod 2579 &\equiv 1061 \bmod 2579 \\ 2^{32} \bmod 2579 &\equiv 1277 \bmod 2579 \\ 2^{64} \bmod 2579 &\equiv 801 \bmod 2579 \\ 2^{128} \bmod 2579 &\equiv 2009 \bmod 2579 \\ 2^{256} \bmod 2579 &\equiv 2525 \bmod 2579 \\ 2^{512} \bmod 2579 &\equiv 337 \bmod 2579 \end{aligned}$$

Damit:

$$\begin{aligned} 2^{765} \bmod 2579 &\equiv (337 \cdot 2009 \cdot 801 \cdot 1277 \cdot 1061 \cdot 256 \cdot 16 \cdot 2) \bmod 2579 \\ &\equiv 949 \bmod 2579. \end{aligned}$$

Damit hat Alice also ihr Schlüsselpaar generiert:

$$\begin{aligned} \mathrm{K}_{pub}^{A} &= [p, g, y] = [2579, 2, 949] \\ \mathrm{K}_{priv}^{A} &= [x] = 765 \end{aligned}$$

④ Nun nehmen wir an, Bob möchte den Klartext `bob` an Alice senden, verschlüsselt mit dem ElGamal Kryptosystem. Dazu benutzt Bob den Public Key von Alice: $\mathrm{K}_{pub}^{A} = [p, g, y] = [2579, 2, 949]$. Er codiert zunächst – das muß auch Alice bekannt sein – den Klartext durch Zahlen, zum Beispiel durch die ASCII Werte der Zeichen

$$\begin{aligned} \texttt{b} &\longrightarrow 98 \\ \texttt{o} &\longrightarrow 111 \\ \texttt{b} &\longrightarrow 98 \end{aligned}$$

also ist die Klartextnachricht $M = 9811198$ zu verschlüsseln. Nun muß gemäß dem Verschlüsselungsverfahren der Klartext a eine Zahl in $\mathbb{Z}_p^* = \mathbb{Z}_{2579}^*$ sein, daher wird M in die beiden Blöcke

$$\begin{aligned} a_1 &= 981 \\ a_2 &= 1198 \end{aligned}$$

aufgeteilt.

⑤ Um den ersten Block zu verschlüsseln, wählt Bob eine zufällige Zahl $k_1 \in \mathbb{Z}_{p-1}$, zum Beispiel $k_1 = 853$. Dann berechnet er:

$$\begin{aligned} y_1^1 &= g^{k_1} \bmod p \\ &= 2^{853} \bmod 2579 \\ &\equiv 435 \bmod 2579 \end{aligned}$$

und

$$\begin{aligned} y_2^1 &= (a_1 \cdot y^{k_1}) \bmod p \\ &= (981 \cdot 949^{853}) \bmod 2579 \\ &\equiv 106 \bmod 2579 \end{aligned}$$

Damit lautet der erste Chiffreblock:

$$C^1 = (y_1^1, y_2^1) = (435, 106)$$

Der zweite Block wird mit einer zweiten zufälligen Zahl $k_2 = 75$ folgendermaßen chiffriert:

$$\begin{aligned} y_1^2 &= g^{k_2} \bmod p \\ &= 2^{75} \bmod 2579 \\ &\equiv 204 \bmod 2579 \end{aligned}$$

und

$$\begin{aligned} y_2^2 &= (a_2 \cdot y^{k_2}) \bmod p \\ &= (1198 \cdot 949^{75}) \bmod 2579 \\ &\equiv 2371 \bmod 2579 \end{aligned}$$

Damit lautet der zweite Chiffreblock:

$$C^2 = (y_1^2, y_2^2) = (204, 2371)$$

Damit ist der an Alice übertragene Chiffretext:

$$C = C^1 C^2 = (435, 106)(204, 2371)$$

⑥ Alice dechiffriert den ersten erhaltenen Chiffreblock

$$C^1 = (y_1^1, y_2^1) = (435, 106)$$

indem sie mit ihrem Private Key $\mathrm{K}_{priv}^{A} = [x] = 765$ zunächst den Wert

$$z = (y_1^x)^{-1} \bmod p$$

berechnet:

$$\begin{aligned} z &= ((y_1^1)^x)^{-1} \bmod p \\ &= (435^{765})^{-1} \bmod 2579 \\ &= (435)^{2579-765-1} \bmod 2579 \\ &= 435^{1813} \bmod 2579 \\ &\equiv 1980 \bmod 2579 \end{aligned}$$

Dann berechnet sie:

$$\begin{aligned} \text{klartextblock}_1 &= y_2^1 \cdot z \bmod p \\ &= 106 \cdot 1980 \bmod 2579 \\ &= 981 \bmod 2579. \end{aligned}$$

Der zweite Chiffreblock

$$C^2 = (y_1^2, y_2^2) = (204, 2371)$$

wird folgendermaßen dechiffriert:

$$\begin{aligned} z &= ((y_1^2)^x)^{-1} \bmod p \\ &= (204^{765})^{-1} \bmod 2579 \\ &= (204)^{2579-765-1} \bmod 2579 \\ &= 204^{1813} \bmod 2579 \\ &\equiv 515 \bmod 2579 \end{aligned}$$

Damit berechnet sie:

$$\begin{aligned}\text{klartextblock}_2 &= y_2^2 \cdot z \bmod p \\ &= 2371 \cdot 515 \bmod 2579 \\ &= 1198 \bmod 2579.\end{aligned}$$

Damit wird der Klartext:

$$\text{klartext} = 9811198 \xrightarrow{\text{ASCII}} \texttt{bob}$$

Wie aus der allgemeinen Darstellung des ElGamal Kryptosystems und den durchgeführten Beispielen zu erkennen ist, erzeugt dieses Verfahren einen Chiffretext, der doppelt so groß ist wie der Klartext. Aus diesem Grund ist die Abbildung $d_{\mathrm{K}_{priv}}$ nicht injektiv und daher gilt im Gegensatz zum RSA Kryptosystem:

$$e_{\mathrm{K}_{pub}}(d_{\mathrm{K}_{priv}})(M) \neq M$$

Das bedeutet also, dass mit dieser Form des ElGamal Verfahrens keine Authentifizierung erzielt werden kann.

Das ElGamal Kryptosystem ist unsicher, wenn Eve den Wert

$$x = \log_g y$$

berechnen kann. Dann kann Eve den Chiffretext in exakt der gleichen Art und Weise dechiffrieren wie Alice. Daher ist eine notwendige Bedingung für die Sicherheit des ElGamal Kryptosystems, dass das diskrete Logarithmusproblem auf $\mathbb{Z}_p^*$ nicht durchführbar ist. Dies ist im allgemeinen der Fall, wenn die Primzahl p geeignet gewählt wird und g eine primitive Wurzel modulo p ist. Es ist für diese Version des diskreten Logarithmusproblems kein Polynomialzeit Algorithmus bekannt, der den diskreten Logarithmus berechnet. Um Angriffen vorzubeugen, sollte p wenigsten 300 Stellen haben und $p-1$ sollte wenigstens einen großen Primfaktor enthalten.

Kapitel 7

Zero-Knowledge Protokolle

7.1 Die grundlegende Idee

Zero-Knowledge Protokolle werden synonym auch mit **Zero-Knowledge Beweise** bezeichnet[1]. Solche Protokolle dienen dem Zweck, jemanden davon zu überzeugen, dass man ein Geheimnis kennt, ohne jedoch das Geheimnis selbst preiszugeben. Bei einem Zero-Knowledge Protokoll kommunizieren also zwei Parteien miteinander. Die eine Partei nennt man *Beweiser*, die andere heißt *Verifizierer*. Der Beweiser überzeugt dabei den Verifizierer mit einer gewissen Wahrscheinlichkeit davon, dass er im Besitz eines Geheimnisses ist, ohne dabei Informationen über das Geheimnis selbst bekannt zu geben. Eine typische Anwendung, in der diese Thematik eine große Rolle spielt, ist die **Teilnehmerauthentifikation**. Hierbei ist der Beweiser beispielsweise ein Benutzer, der sich in einem Netzwerk anmeldet, der Verfizierer ist ein Server.

Die Idee der Zere-Knowledge Protokolle stammt von SHAFI GOLDWASSER, SILVIO MICALI und CHARLES RACKOFF [98]. Ein **interaktiver Zero-Knowledge Beweis** ist folgendermaßen charakterisiert:

- ☞ Wenn eine Behauptung richtig ist und Alice einen Beweis dafür kennt, dann kann sie Bob auf jeden Fall von der Richtigkeit der Behauptung überzeugen. Dies nennt man **Durchführbarkeit** des Beweises.
- ☞ Wenn die Behauptung nicht richtig ist oder Alice kennt keinen Beweis dafür, so läßt sich Bob nur mit einer sehr geringen Wahrscheinlichkeit überzeugen (**Korrektheit**).
- ☞ Der Beweis hat die **Zero-Knowledge Eigenschaft**, das bedeutet, Bob gewinnt während des Beweises kein Wissen hinzu außer, dass Alice einen Beweis für die Behauptung kennt.

  Formal wird diese Eigenschaft folgendermaßen beschrieben: Es gibt einen **Simulator**, der ohne Kenntnis des Beweises einen interaktiven Beweis konstruieren kann. Dieser konstruierte Beweis ist für einen Außenstehenden nicht von einem echten inteaktiven Beweis zu unterscheiden.

[1] Die tiefgehendste Diskussion von Zero-Knowledge Protokollen in der Literatur findet man in [97, Chap. 4]. Einen Überblick über Zero-Knowledge Protokolle findet man in dem Artikel von IAN STEWART in [136].

Um diese Definition zu veranschaulichen betrachten wir zwei klassische Beispiele.

Beispiel[2]:

Der italienische Mathematiker NICCOLO TARTAGLIA entdeckte im Jahre 1535 die Formel zur Lösung der allgemeinen kubischen Gleichung

$$ax^3 + bx^2 + cx + d = 0. \tag{7.1}$$

Das heißt, TARTAGLIA war in der Lage, eine geschlossene Form für die Nullstellen des allgemeinen Polynoms 3. Ordnung anzugeben. Um nun seinen Fachkollegen zu zeigen, dass er die Formel zur Lösung kannte, verwendete TARTAGLIA einen Zero-Knowledge Beweis. Das bedeutet, er wollte seine Kollegen davon überzeugen, dass er die Formel kannte ohne jedoch sein Geheimnis zu verraten. Dazu ließ er sich von seinen Kollegen eine Reihe kubischer Formeln geben (i.e. eine Reihe von Gleichungen der Form (7.1) mit verschiedenen Koeffizienten a, b, c, d), löste diese unbeobachtet und teilte seinen Kollegen die Lösung mit. Durch Einsetzen konnten diese dann einfach verifizieren, dass die Lösung korrekt war.

Beispiel:

Das folgende Beispiel geht auf JEAN-JACQUES QUISQUATER [173] zurück.

Man betrachtet ein Gebäude, das aus einem Vorraum besteht (siehe Abbildung [7.1]), von dem aus zwei Zimmer zu erreichen sind. Die beiden Zimmer sind nun durch eine magische Tür verbunden. Diese magische Tür läßt sich nur durch ein geheimes Passwort öffnen.

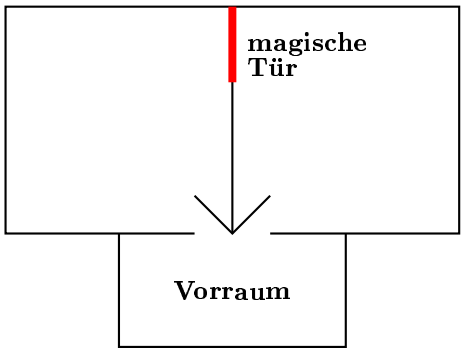


Abbildung 7.1: Die magische Tür

Alice behauptet nun Bob gegenüber, das geheime Passwort zu kennen und will dies Bob beweisen.

[2]Siehe dazu auch [71, Chap. 6].

❶ Eine Möglichkeit besteht darin, dass Alice und Bob zu der magischen Tür gehen. Alice öffnet die magische Tür und damit ist Bob davon überzeugt, dass Alice das Passwort kennt. Dies ist aber kein Zero-Knowledge Beweis, da unklar ist, ob Bob nicht doch Wissen hinzugewonnen hat, er könnte etwas über die Länge des Passworts erfahren haben usw.

❷ Um einen Zero-Knowledge Beweis auszuführen, verfahren Alice und Bob folgendermaßen. Alice betritt das Gebäude allein und geht in eines der beiden Zimmer. Anschließend betritt Bob den Vorraum. Er kann nicht sehen, in welchem Zimmer sich Alice befindet. Nun wählt Bob aus – beispielsweise indem er eine Münze wirft – aus welchem Raum Alice den Vorraum betreten soll. Wenn er laut *Rechts* ruft, soll Alice den Vorraum aus dem rechten Zimmer betreten, ruft er *Links*, dann muss sie vom linken Zimmer aus den Vorraum betreten.

Dieses Szenario wird mehrere Mal wiederholt. Wenn Alice tatsächlich das Passwort kennt, hat sie keinerlei Schwierigkeiten, Bobs Aufforderung nachzukommen und den Vorraum aus den geforderten Zimmer zu betreten. Kennt sie das Passwort nicht, muss sie in jeder Runde raten, welches Zimmer Bob wählen wird. In jeder einzelnen Runde ist die Wahrscheinlichkeit, dass Alice richtig rät $\frac{1}{2}$. Werden n Runden durchgeführt, dann beträgt die Wahrscheinlichkeit, dass Alice jedesmal richtig tippt, $\frac{1}{2^n}$.

Diesen Beweis kann man auch durch eine Simulation darstellen. Anstelle von Alice betritt Erna das Gebäude. Erna kennt das Passwort für die magische Tür nicht. Bob nimmt diesmal eine Videokamera mit in den Vorraum und zeichnet damit alle Vorgänge inklusive seiner Wahl auf. Bob wirft wieder seine Münze und wählt eine Tür. Kommt Erna aus der richtigen Tür, beginnen beide eine neue Runde. Kommt Erna aus der falschen Tür, dann löscht Bob die Aufzeichnung. Dies wird solange durchgeführt, bis n richtige Szenen im Kasten sind. Durchschnittlich brauchen Bob und Erna doppelt so viele Durchläufe wie Alice und Bob.

Der Videofilm zeigt dann einen interaktiven Beweis der Behauptung *Erna kennt das Passwort.* Dieser Film ist von einem neutralen Beobachter nicht von einem zweiten Film zu unterscheiden, der von dem Szenario mit Alice gemacht wurde. Erna kennt das Passwort aber nicht. In dem Film mit Erna - und damit in dem interaktiven Beweis – kann also keinerlei Information über das Passwort enthalten sein. Bob und Erna haben nach dem Video also nicht mehr Informationen als vorher.

7.2 Das Fiat-Shamir Protokoll

Eines der bekanntesten Beispiele für einen Zero-Knowledge Beweis ist das **Fiat-Shamir Protokoll**, das im Jahre 1986 von AMOS FIAT und ADI SHAMIR entwickelt wurde.

Abbildung 7.2: Adi Shamir

Ausgangspunkt des FIAT-SHAMIR Protokolls sind wieder zwei ausreichend große Primzahlen p, q, so dass $n = p \cdot q$ schwierig zu faktorisieren ist. Die beiden zahlentheoretischen Grundlagen des FIAT-SHAMIR Protokolls sind (siehe dazu Kapitel [17.3]):

- Sei eine Zahl $a \in \mathbb{Z}_n$ gegeben, $a \neq 0$. Die Gleichung

$$x^2 \bmod n \equiv a$$

  hat entweder vier Lösungen (i.e. vier x-Werte) oder keine Lösung. Im ersten Fall heißt a **quadratischer Rest**, im zweiten Fall **quadratischer Nichtrest**.

- Das Berechnen von diskreten Quadratwurzeln ist äquivalent zur Faktorisierung von n, i.e. man kann genau dann die Quadratwurzeln eines quadratischen Rests modulo n bestimmen, wenn man n faktorisieren kann (siehe dazu Kapitel [17.4]).

Mit dem FIAT-SHAMIR Protokoll kann jemand die Behauptung beweisen, dass eine Zahl v ein quadratischer Rest ist und dass er eine Quadratwurzel modulo n kennt.

① **Szenario:**
Alice will sich Bob gegenüber authentifizieren, i.e.

② **Schlüsselerzeugung:**

- Wähle zwei zufällige große Primzahlen p, q, bilde $n = p \cdot q$.

- Alice wählt eine zufällige Zahl s, $1 < s < n$. s ist nur Alice bekannt.
- Alice berechnet

$$v = s^2 \bmod n$$

Damit hat Alice das Schlüsselpaar generiert: $K_{pub} = v$, $K_{priv} = s$.

③ **Authentifizierung:**
Das Authentifizierungsprotokoll wird folgendermaßen durchgeführt. Alice behauptet, für eine Zahl v eine Quadratwurzel zu kennen[3]. Dazu veröffentlicht Alice v zusammen mit ihrem Namen und hält den Wert s geheim. Angenommen Alice kennt tatsächlich eine Zahl s mit $s^2 \bmod n = v$. Um Bob davon zu überzeugen – ohne den Wert s preiszugeben – führt Alice mit Bob das folgende Protokoll durch:

- **Commitment:** Alice wählt eine zufällige Zahl r und berechnet

$$x = r^2 \bmod n$$

Alice sendet den Wert x an Bob.

- **Challenge:** Bob wählt zufällig ein Bit b, also `0` oder `1`. Bob sendet diesen Wert b an Alice.
- **Response:** Alice berechnet

$$y = \begin{cases} rs \bmod n & \text{falls } b = 1 \\ r \bmod n & \text{falls } b = 0 \end{cases}$$

Alice sendet y an Bob.

- **Verifikation:** Bob überprüft die Antwort von Alice:
Falls $b = 1$:

$$y^2 \bmod n = xv \bmod n$$

Falls $b = 0$:

$$y^2 \bmod n = x.$$

Beispiel:

Schlüsselerzeugung:

Alice wählt die Primzahlen $p = 7, q = 23$ und berechnet $n = p \cdot q = 161$.

Weiterhin wählt Alice ein zufälliges s, $1 < s < n$, z.B. $s = 31$. Sie berechnet

$$v = s^2 \bmod n = 31^2 \bmod 161 = 156.$$

Damit ist:

$$K_{\text{pub}} = (v, n) = (156, 161) \qquad K_{\text{priv}} = (s, n) = (31, 161)$$

Der Public-Key $K_{\text{pub}} = (v, n) = (156, 161)$ wird an Bob gesendet.

Protokoll

[3] Also: Alice behauptet, sie kennt die Zahl s, die die Eigenschaft $s^2 \bmod n = v$ hat.

- **Commitment:** Alice wählt die Zahl $r = 17$ und berechnet

$$x = r^2 \bmod n = 17^2 \bmod 161 = 128.$$

  x wird an Bob gesendet.

- **Challenge:** Bob wählt $b = 1$ und sendet den Wert an Alice.

- **Response:** Alice berechnet

$$y = r \cdot s \bmod n = 17 \cdot 31 \bmod 161 = 44$$

  und sendet $y = 44$ an Bob.

- **Verifikation:** Bob berechnet einerseits den Wert:

$$y^2 \bmod n = 44^2 \bmod 161 = 4.$$

  Andererseits berechnet er:

$$(x \cdot v) \bmod n = 128 \cdot 156 \bmod 161 = 4.$$

Ähnlich wie bei dem Beispiel der magischen Tür kann Alice in diesme Protokoll betrügen. Wenn Alice von Bob das Bit $b = 0$ erhält, kann sie natürlich das Protokoll unverändert durchführen, da sie in diesem Fall die Quadratwurzel (i.e. den Wert s) nicht benötigt. Vermutet Alice, dass Bob den Wert $b = 1$ sendet, dann sendet sie in der Commitment-Phase den Wert $x \cdot v^{-1}$ und als Respond den Wert r. Die Verifikation gelingt in diesem Fall, da

$$r^2 \equiv x \cdot v \equiv r^2 \cdot v^{-1} \cdot v \bmod n$$

Mit anderen Worten: Bob muß die Wahl von b daher wirklich zufällig treffen, so dass sie für Alice unvorhersehbar ist. Dieser Betrug gelingt daher mit einer Wahrscheinlichkeit $1/2$.

Aus diesem Grund müssen Alice und Bob dieses Protokoll (i.e. die Schritte von der Commitment-Phase bis zur Verifikation) mehrmals durchführen. Dies geschieht so lange, bis Bob davon überzeugt ist, dass Alice tatsächlich den Wert s kennt, i.e. eine Quadratwurzel von v mod n. Nach n Runden hat Alice nur noch eine Betrugswahrscheinlichkeit von 2^{-n}. Bob kann daher durch die Anzahl der Runden das Sicherheitsniveau nach Belieben hochsetzen.

Durchführbarkeit

Das FIAT-SHAMIR Protokoll ist tatsächlich ein Zero-Knowledge Beweis für die Behauptung, Alice kennt eine Quadratwurzel von v mod n. Denn kennt Alice wirklich die Quadratwurzel, kann sie immer richtig antworten. Damit ist die Eigenschaft der Durchführbarkeit erfüllt.

Korrektheit

Wenn Alice die Quadratwurzel nicht kennt, so kann sie nicht beide möglichen Fragen von Bob – die Werte $b = 0$ oder $b = 1$ – beantworten. Wenn Alice

dies dennoch könnte – also die Quadratwurzel nicht kennt und die Frage richtig beantworten – so könnte sie Quadratwurzeln berechnen, denn der Quotient der beiden Antworten $r \cdot s/r$ ist eine Quadratwurzel von v. Dies ist jedoch nicht möglich, da die Berechnung von Quadratwurzeln modulo n genau so schwierig ist wie die Zahl n zu faktorisieren. Daher ist der Beweis korrekt.

Zero-Knowledge Eigenschaft

Um zu zeigen, dass das FIAT-SHAMIR Protokoll die Zero-Knowledge Eigenschaft hat, muss man einen Simulator angeben. Der Simulator arbeitet folgendermaßen:

- Der Simulator wählt eine Zahl r zufällig aus und bestimmt ein Bit $c \in \{0, 1\}$. Er berechnet

  $$x = r^2 \cdot v^{-c} \bmod n$$

  und sendet x an Bob.

- Bob wählt ein Bit $b \in \{0, 1\}$.

- Ist $b = c$, dann kann der Simulator korrekt antworten. Dazu schickt er r an Bob, denn es gilt

  $$r^2 \bmod n = r^2 \cdot v^{-b} \cdot v^b \bmod n = r^2 \cdot v^{-c} \cdot v^b \bmod n = x \bmod n$$

  Wenn $b \neq c$, dann wird die Runde gelöscht.

7.3 Zero-Knowledge Beweis für die Kenntnis eines diskreten Logarithmus

Ein weiteres Beispiel eines Zero-Knowledge Beweises ist der Beweis, einen diskreten Logarithmus zu kennen. Wie beim ElGamal Kryptosystem (siehe Kapitel [6.6]) seien eine Primzahl p und eine Zahl $g \in \mathbb{Z}_p^*$ gegeben, so dass die Berechnung des diskreten Logarithmus zur Basis g mod p nicht effizient möglich ist. Die Behauptung von Alice lautet nun: *Zu einem y kenne ich einen diskreten Logarithmus x, das heißt eine Zahl x mit*

$$y \equiv g^x \bmod p$$

Für ein Zero-Knowledge Protokoll, basierend auf dem diskreten Logarithmusproblem gehen Alice und Bob folgendermaßen vor:

- **Schlüssel:** y, g, p sind öffentlich, der Exponent x ist nur Alice bekannt.
- **Commitment:** Alice wählt eine zufällige Zahl r und berechnet

  $$k = g^r \bmod p.$$

  Alice sendet den Wert k an Bob.
- **Challenge:** Bob wählt zufällig ein Bit b, also `0` oder `1`. Bob sendet diesen Wert b an Alice.
- **Response:** Alice berechnet

  $$y = (r + bx) \bmod p.$$

  Alice sendet y an Bob.
- **Verifikation:** Bob überprüft die Antwort von Alice:
  Falls $b = 0$ püft Bob, ob

  $$g^r \bmod p = k$$

  gilt. Falls $b = 1$ wird geprüft, ob

  $$g^{r+x} \bmod p = k \cdot y \bmod p$$

  gilt.

Beispiel:

Kapitel 8

Digital Cash

Angenommen, ein Abgeordneter XY erhält großzügige Geldgeschenke aus irgendwelchen dunklen Kanälen[1]. Aus offensichtlichen Gründen möchte er diese Transaktion verheimlichen und anstelle dessen vorgeben, dass das Geld von einer legitimen Spende herrührt. Oder man betrachte das Szenario, dass ein großzügiger Spender nicht möchte, dass unser Abgeordneter XY weiß, von wem das Geld stammt. Wenn nun der Spender mit einem Scheck bezahlt, dann können geeignet informierte Mitarbeiter der involvierten Bank ihn verraten. Aus ähnlich gelagerten Gründen ist es dem Abgeordneten XY nicht möglich, Bezahlungen via Kreditkarte zu erhalten. Mit anderen Worten, das einzige anonyme Bezahlungsschema scheint Bargeld zu sein.

Nehmen wir nun an, dass unser Abgeordneter XY bereits seit mehreren Legislaturperioden seinen Dienst versieht und wir nähern uns dem Ende des 21. Jahrhunderts. Die Abwicklung der Wirtschaft geschieht komplett in elektronischer Form. Die sich nun erhebende Frage ist, ob es eine elektronisches Form des klassischen Bargelds geben kann. Dabei treten eine Reihe von Problemen auf.

Beispielsweise war es Anfang des 21. Jahrhunderts möglich, Geldscheine in solcher Qualität zu photokopieren, dass eine Inspektion recht schnell feine Unterschiede zwischen Kopie und Original feststellen konnte. Kopien von elektronischen Informationen sind jedoch nicht vom Original zu unterscheiden. Aus diesem Grund kann eine Person, die eine gültige elektronische Münze besitzt, im Prinzip beliebig viele Kopien herstellen. Daher benötigt man eine Methode, die die doppelte Geldausgabe verhindert. Eine Möglichkeit dies umzusetzen, wäre eine Bank, die ein Verzeichnis unterhält, in der jede Münze aufgeführt ist und wer sie gerade besitzt. Wenn jedoch Datensätze über Münzen existieren, die auch die Besitzerwechsel mit aufzeichnen, dann ist die Anonymität nicht mehr gewährleistet. Gelegentlich kann es zu zeitweisen Ausfällen der Kommunikation mit der Zentralbank kommen. Dann hat unter Umständen derjenige, der das Geld erhält das Problem, dass er nicht mehr die Gültigkeit des Besitztums der Münze bei der Bank hinterlegen kann.

In [78, p. 129 f.] werden eine Reihe von Eigenschaften elektronischer Münzen

[1] Dieser Abschnitt lehnt sich sehr eng an die Darstellung in [215, Chapter 9] an. Siehe auch [22], [78], [218, Kapitel 10.5] und Kapitel 6.4 in SCHNEIERs Buch [193].

angegeben, die sich aus Eigenschaften der klassischen Bargeldes ableiten. Dies sind (unter anderem):

1. Das Bezahlen muß einfach und mit sehr geringen Kosten erfolgen. Damit wird elektronisches Bargeld auch für sehr kleine Beträge anwendbar (Micro-Payment).

2. Das Bezahlen von kleinen Beträgen muß unverbindlich sein, das bedeutet, der Kunde möchte bein Kauf einer Zeitung keine Transaktion abwickeln, die er später auf seinem Kontoauszug wiederfindet und kontrollieren muß.

3. Das Bezahlen soll nicht nur bei Händlern sondern auch zwischen Privatpersonen möglich sein.

4. Das elektronische Bargeld sollte möglichst viele Sprünge machen können bevor es wieder an die bank zurückkommt.

In der Arbeit [163] listen T. Okamoto und K. Ohta weitere Eigenschaften auf, die ein elektronisches Bargeldsystem aufweisen muß:

1. **Unabhängigkeit**
   Die Sicherheit des elektronischen Geldes darf nicht von irgendwelchen physikalischen Randbedingungen abhängen. Das elektronische Bargeld kann sicher durch Computernetzwerke übertragen werden.

2. **Sicherheit**
   Das elektronische Bargeld kann nicht kopiert und mehrfach verwendet werden. Eine Fälschung des Geldes muß verhindert werden.

3. **Geheimhaltung** (privacy, untraceability)
   Derjenige, der das Bargeld ausgibt, kann anonym bleiben. Falls das Bargeld in legitimer Weise ausgegeben wird, kann weder der Empfänger der Zahlung noch die Bank den Auszahler identifizieren.

4. **Off-line Bezahlung**
   Die Transaktion kann *offline* geschehen. Wenn ein Benutzer mit elektronischem Geld in einem Shop bezahlt, kann diese Transaktion zwischen Benutzer und Shop ohne Zugriff auf eine zentrale Bank geschehen. Das bedeutet also, es ist keine Kommunikation mit der zentralen Bank notwendig, um die Transaktion auszuführen.

5. **Übertragbarkeit**
   Das elektronische Bargeld kann zu anderen Personen transferiert werden.

6. **Teilbarkeit**
   Ein Betrag an elektronischem Bargeld kann in kleinere Beträge aufgeteilt werden.

Okamoto und Ohta geben ein System an, das alle sechs Kriterien erfüllt. Verschiedene Geldsysteme, die einige dieser Anforderungen erfüllen, wurden von David Chaum designed[2]. In Kapitel [8.4] beschreiben wir ein System, das die Kriterien 1 bis 4 erfüllt. Dieses Protokoll wurde von Stefan Brands entwickelt [30].

[2] Siehe die Arbeiten [41–43]

Offensichtlich sind die Verfahren des Bezahlens mit elektronischen Münzen beträchtlich komplexer als das jahrhundertealte Bezahlsystem mit realen Münzen. Der Grund liegt darin, dass im Gegensatz zu physikalischen Objekten elektronische Objekte ohne großen Aufwand reproduziert werden können. Daher sind eine Reihe zusätzlicher Schritte erforderlich, elektronische Geldfälscher dingfest zu machen. Dies impliziert, dass im Prinzip die digitale Signatur des Eigentümers an eine elektronische Münze angehängt werden muß. Dies wiederum hat natürlich den Effekt, dass die Anonymität nicht gewährleistet ist. Die Lösung dieses Problems – entwickelt von DAVID CHAUM – nennt man **blinde Signatur**. Dieser Prozess trägt einiges zur Komplexität der Protokolle bei.

8.1 Elektronisches Bargeld

Das Aufkommen von Electronic Commerce – i.e. beim Bezahlen von Waren, Dienstleistungen oder Informationen, die im Internet angeboten werden – hat zu neuen Problemstellungen im Umfeld der Sicherheit geführt. Die Kosten für viele dieser Dienste bewegen sich im Bereich von wenigen Cents (Micro-Payment). Daher ist eine bargeldlose Transaktion wie z.B. eine Überweisung oder die Belastung einer Kreditkarte unrentabel. Vielfach möchte der Kunde für die einmalige oder seltene Nutzung eines Dienstes keine persönlichen Daten oder Kontodaten angeben. Für solch ein Szenario bietet sich das Bezahlen mit **elektronischen Münzen** an. Der eigentliche Bezahlvorgang besteht aus der Übertragung von einigen elektronischen Münzen – i.e. Bitfolgen – zwischen Kunde und Händler. Wie bei einer klassischen Bargeldtransaktion werden zwischen den beiden Parteien Objekte – hier sind das elektronische Münzen – ausgetauscht. Wie beim klassischen Bargeld sollte dieser Vorgang anonym erfolgen, gleichzeitig aber auch sicher gegen Betrug sein.

Die Bezahlung mit den elektronischen Münzen sicher und effizient zu gestalten ist eine Aufgabe der modernen Kryptographie. Wir werden nun schrittweise ein Protokoll vorstellen, das von DAVID CHAUM (siehe Abbildung [8.1]) entwickelt und patentiert wurde. DAVID CHAUM ist Gründer der niederländischen Firma Digicash.

Abbildung 8.1: DAVID CHAUM

Die an den dargestellten Verfahren beteiligte Bank nennen wir E-Bank, als Zahlungsmittel werden elektronische Münzen verwendet, diese nennen wir E-

Münze. Die Person, die die elektronische Münze verwenden und damit einkaufen will, ist Alice. Sie hat es mit einem Händler zu tun. Unter einer elektronischen Münze hat man sich eine endliche Bitfolge vorzustellen.

- **Protokoll 1**

  1. Alice beantragt bei ihrer Bank eine E-Münze.
  2. Die E-Bank generiert darauf hin in ihrem Rechenzentrum eine Datei mit dem Inhalt

     E-Münze; Wert: 5 €,

     belastet Alices Konto mit diesem Betrag und transferiert die E-Münze auf Alices PC.
  3. Alice geht mit der E-Münze einkaufen, bezahlt damit ihre Ware bei einem Händler.
  4. Der Händler reicht die E-Münze bei irgendeiner Bank ein (er hat ja keinerlei Kenntnis darüber, bei welcher Bank Alice die E-Münze beantragt hat) um sie auf seinem Konto gutschreiben zu lassen.

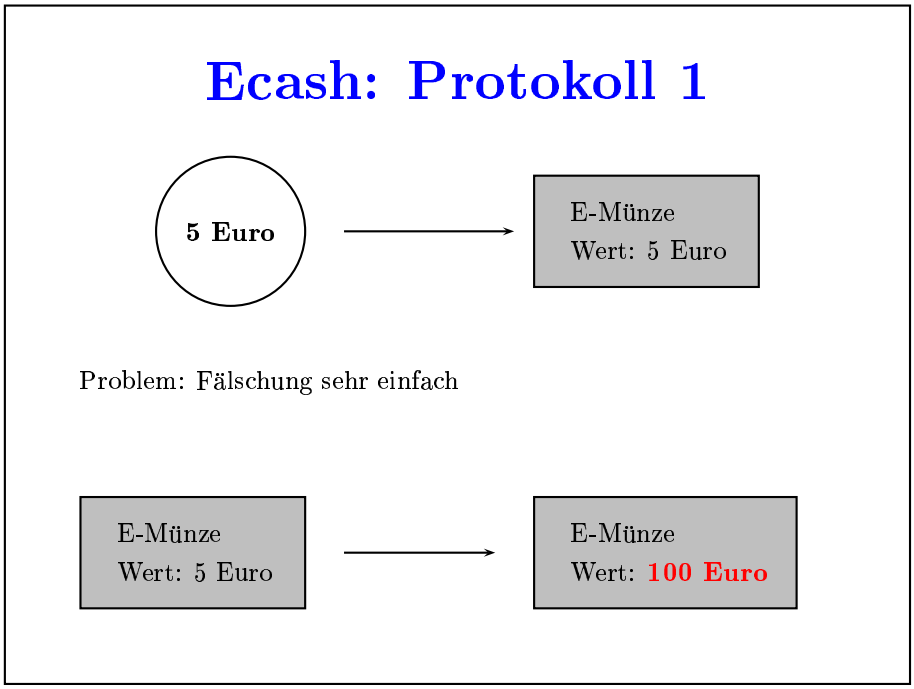


Abbildung 8.2: Einfaches Protokoll zur Erzeugung einer E-Münze

Dieses Verfahren ist in der Abbildung [8.2] dargestellt. Diese Form der Erzeugung eletronischen Bargelds führt unmittelbar zur Inflation, denn es ist in diesem Verfahren kein Mechanismus eingebaut, der verhindert, dass derjenige, der die Münze ausgibt, den Wert der Münze beliebig ändert.

Eine weitere Möglichkeit des betrugs besteht darin, dass Alice – ehrlich wie sie nun mal ist – mit einer 5 € E-Münze beim Händler bezahlt, dieser aber die Münze als 10 € Münze beid er bank einreicht.

Das Protokoll enthält auch kein Mechanismus, das Kopieren einer Münze zu verhindern. Dieser Betrug kann sowohl von Alice als auch dem Händler durchgeführt werden. Zusammengefasst kann man konstatieren: Protokoll 1 gewährleistet die Anonymität von Alice – sowohl gegenüber der Bank als auch gegenüber dem Händler. Es gewährleistet jedoch keinerlei Schutz gegenüber Betrug, in Form von Änderung des Wertes als auch Mehrfachausgabe der Münze. Die Bank, die die E-Münze wieder einlösen soll, hat auch keinerlei Gewährleistung, ob diese E-Münze echt ist oder ob sie der Händler auf seinem PC generiert hat.

- **Protokoll 2**
  Um die Änderung des Wertes einer elektronischen Münze zu verhindern, versieht die Bank jede Münze mit einer Unterschrift (i.e. digitalen Signatur), die nur die Bank und niemand sonst erstellen kann. Durch diesen Mechanismus wird verhindert, dass der Kunde, der die elektronische Münze von der Bank erhält und auf seinem Rechner speichert, den Betrag abändert. Versucht er dennoch den Betrag zu manipulieren, wird durch die digitale Signatur dieser Betrugsversuch erkannt und die digitale Signatur ist damit ungültig. Dieses Protokoll ist in der Abbildung [8.3] schematisiert.

  Digitale Signaturen haben im Gegensatz zu herkömmlichen Unterschriften die Eigenschaft, dass sie sowohl den Urheber als auch das unterzeichnete Dokument eindeutig identifizieren. Dies hat zur Folge, dass ein Dokument M auf einen eindeutigen Wert $H(M)$ führt. Wird der Inhalt des Dokumentes M zu M' geändert, so ergibt eine erneute Anwendung des Signaturverfahrens auf das manipulierte Dokument (dies ist die Überprüfung) einen Wert $H(M') \neq H(M)$, wodurch die Manipulation festgestellt werden kann. Dieses Protokoll verhindert in der Tat die Manipulation der Daten[3], verhindert aber nicht, dass der Kunde eine beliebige Anzahl von Kopien der digitalen Münze erzeugt und die Münze damit mehrfach ausgibt.

- **Protokoll 3**
  Um das beliebige Kopieren der elektronischen Münzen zu unterbinden, versieht die E-Bank jede Münze mit einer eindeutigen Seriennummer und signiert den kompletten digitalen Inhalt der Münze, insbesondere den Betrag und die Seriennummer[4]. Der Ablauf dieser Protokollvariante ist in der Abbildung [8.4] skizziert. Versucht nun jemand, Kopien der E-Münze zu erzeugen, wird der Betrug von der Bank erkannt.

[3] Genauer, dieses Protokoll gewährleistet die Integrität der auf der elektronischen Münze hinterlegten Daten.

[4] In realen Implementierungen werden auf der elektronischen Münze weitere Informationen gespeiechert, beispielsweise Name der Bank, Erstelldatum und ähnliches. Wir beschränken uns hier in dem Spielmodell auf die wesentlichen Daten.

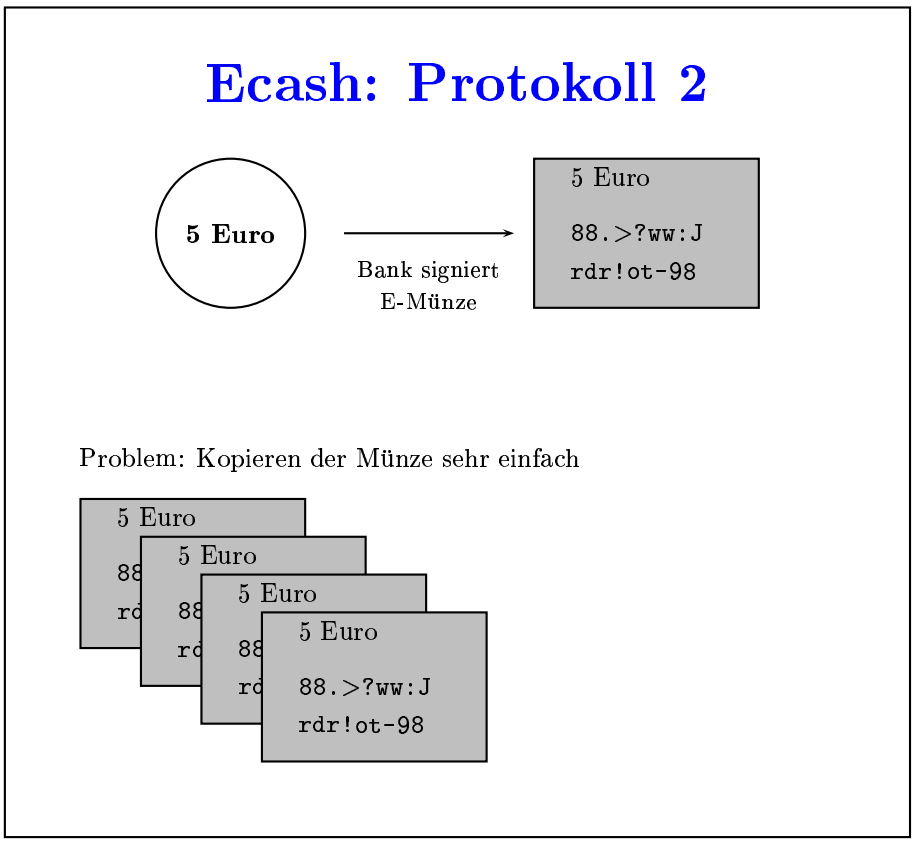


Abbildung 8.3: Protokoll zur Erzeugung einer E-Münze mit digitaler Signatur der Bank

Zu diesem Zweck protokolliert die E-Bank in einer Datenbank alle einlaufenden Seriennummern, beispielsweise in einer Tabelle wie in [8.1] skizziert. Sobald zwei Münzen mit der gleichen Seriennummer zur Bank zurückkommen, ist es an der Zeit, den Staatsanwalt zu informieren.

Das Protokoll 3 ist gegenüber Betrugsversuchen wie Ändern des Inhalts der E-Münze und Mehrfachverwendung sicher. Es zeigt aber noch eine entscheidende Schwäche, denn die **Anonymität** desjenigen, der die E-Münzen ausgibt, ist nicht gewährleistet. Die Bank kann aufgrund der Kunden-Registrierung und Speicherung der Seriennummern ein perfektes Profil jedes Kunden erstellen, was der Ausschnitt in der Tabelle [8.1] andeutet. Das Problem wird offensichtlich durch die Speicherung der Seriennummern der E-Münzen verursacht, auf die man aus Sicherheitsgründen jedoch nicht verzichten kann.

- **Protokoll 4**
  Den Ausweg aus dieser Problematik lieferte David Chaum mit den von ihm entwickelten **blinden Signaturen**. Zu diesem Zweck erzeugt nun der Kunde seine E-Münzen selbst. Um eine gültige 5 Euro E-Münze zu erhalten, generiert der Kunde (auf seinem PC) 100 Dateien. In jeder dieser

Nr.	Ausgabe	Kunde	Kto-Nr.	Händler	Rückl.	Betrag
12345	15.05.2008	Mayr	4711	amazon.de	20.05.2008	30 €
12346	15.05.2008	Mayr	4711	amazon.de	20.05.2008	4 €
12347	15.05.2008	Müller	0815	ARAL	20.05.2008	50 €
12348	15.05.2008	Müller	0815	ARAL	20.05.2008	10 €
12349	15.05.2008	Müller	0815	ARAL	20.05.2008	5 €
12350	18.05.2008	Schmidt	3211	Tchibo	22.05.2008	100 €
⋮		⋮		⋮		⋮

Tabelle 8.1: Möglicher Aufbau eine Datenbank von Transaktionen der Kunden einer E-Bank

Dateien wird der Text *5 Euro* und eine zufällige, große Seriennummer eingetragen. Die Seriennummer muß so groß gewählt werden[5], dass die Wahrscheinlichkeit für das Erzeugen zweier gleicher Nummern beliebig klein ist.

Nun bittet der Kunde die Bank, eine dieser hundert Münzen zu signieren, und zwar so, dass die Bank weder den Betrag noch die Seriennummer kennt. Die Bank wird natürlich nur dann ein Dokument eines Kunden blind signieren, wenn sie sicher ist, dass beispielsweise der Betrag auf der E-Münze 5 Euro ist. Dieses eigentlich widersprüchliche Prozedere wird dadurch realisiert, dass die Bank aus den 100 Dateien zufällig 99 auswählt, die der Kunde offenlegen muß. Wenn die Bank 99 mal von 100 Fällen feststellt, dass der Betrag der E-Münze 5 Euro lautet, dann ist die Wahrscheinlichkeit, dass auf der noch verbleibenden, verdeckten E-Münze ein anderer Betrag steht, äußerst gering. Mit anderen Worten, die Bank kann die letzte Münze blind signieren. Das hierzu verwendete Verfahren betrachten wir in Abschnitt [8.2.1].

Eine Analogie dieses Verfahrens mit Papiergeld besteht aus folgendem Prozedere (siehe Abbildung [8.5]). Der Kunde hat hundert 5 Euro Scheine hergestellt. Jeder Schein hat eine andere Seriennummer und auf jedem Schein ist der Betrag von 5 Euro festgehalten. Der Kunde packt jeden Schein in einen separaten Umschlag zusammen mit einem Blatt Kohlepapier und verschließt den Umschlag. Der Kunde gibt die 100 Umschläge an die Bank, welche 99 zufällig gewählte Umschläge wieder öffnet und deren Inhalt offenlegt. Da von den 100 eingereichten, verdeckten Scheinen 99 in Ordnung sind, geht die Bank davon aus, dass der verbliebene Schein in dem Umschlag ebenfalls korrekt erstellt ist. Um den letzten Umschlag zu signieren, holt die Bank ihren Stempel aus dem Tresor und stempelt den Geldschein durch den Umschlag. Das Kohlepapier hinterläßt dann auf dem Schein den Stempelabdruck. Der Bank-Kunde erhält den gestempelten, i.e. signierten Geldschein zurück, packt diesen aus und kann damit Einkaufen gehen.

Der Kunde geht zu einem Händler, kauft einen Artikel und bezahlt mit

[5] Beispielsweise eine 100stellige Zahl.

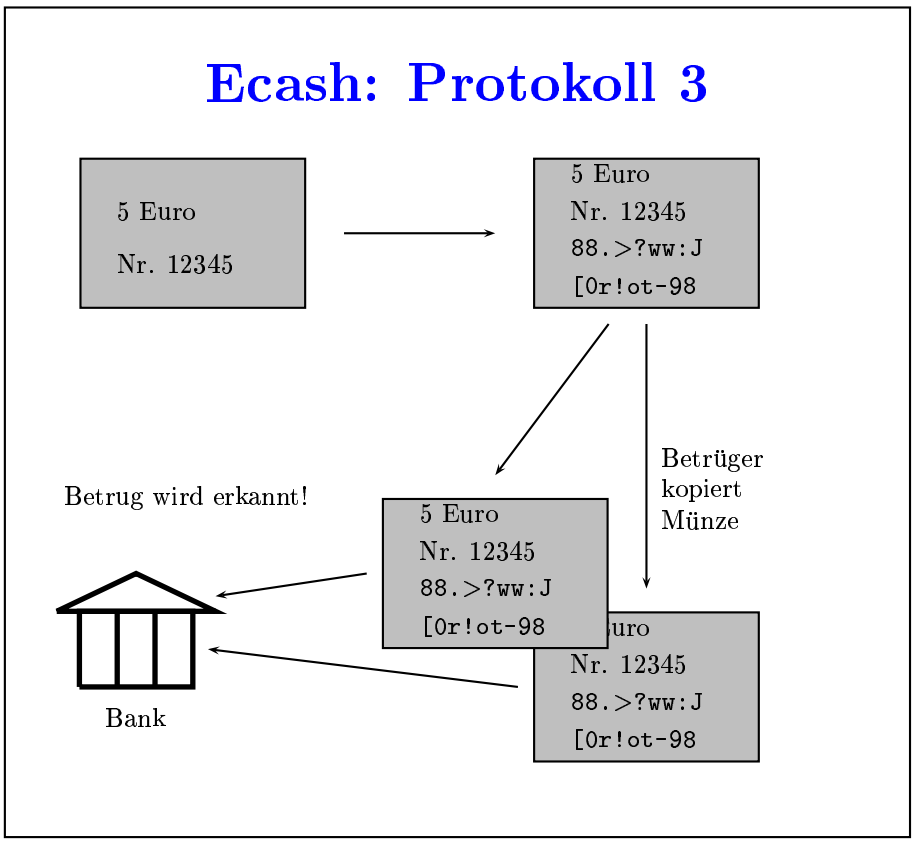


Abbildung 8.4: E-Münze mit Seriennummer und digitaler Signatur; damit wird die Mehrfachverwendung der E-Münze ausgeschlossen

dem (von der Bank) blind signierten Geldschein. Der Händler reicht später[6] diesen Geldschein bei der Bank ein um den Wert des Geldes auf sein Konto gutschreiben zu lassen. Der Händler erkennt anhand der Signatur, welche Bank das Geld freigeschaltet hat. Die Bezahlung einer Ware oder einer Dienstleistung mit diesem Geld hat also die Eigenschaft, dass die Bank nicht nachvollziehen kann, *wer* das Geld ausgegeben hat.

Der Kunde oder der Händler kann nun versuchen, das Geld zu kopieren und das Geld mehrfach auszugeben, bzw. bei der Bank einzureichen. Die Bank erkennt jedoch diesen Betrugsversuch, da die Seriennummern aller einlaufenden Geldscheine registriert werden und jede einlaufende Nummer mit den Beständen abgeglichen wird. Tritt eine Nummer zweimal auf, wird der Betrug erkannt.

Mit anderen Worten, das Protokoll garantiert also die **Anonymität des Kunden** (sowohl gegenüber der Bank als auch gegenüber dem Händler) und bietet **Sicherheit gegenüber Betrugsversuchen**.

[6] Daher ist diese Form der Transaktion *offline*, denn der Händler braucht keinen direkten Kontakt zu Bank.

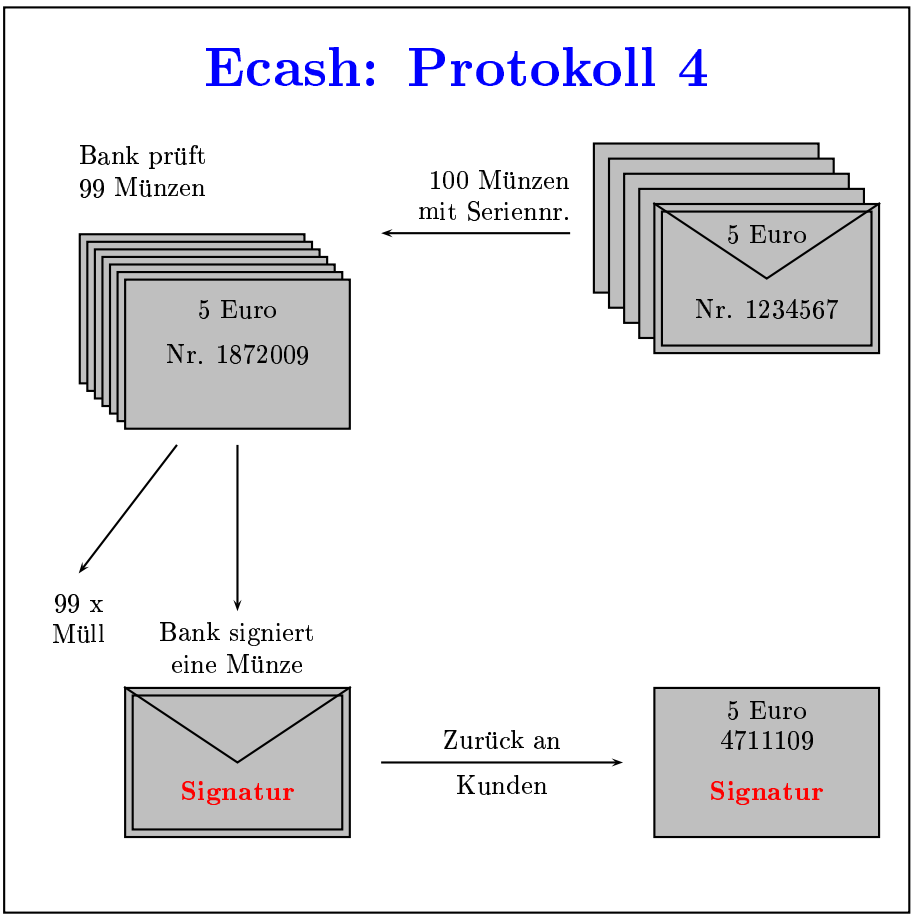


Abbildung 8.5: E-Münze mit Seriennummer und blinder digitaler Signatur; damit wird die Mehrfachverwendung der E-Münze ausgeschlossen und die Anonymität des Kunden ist gewährleistet.

Ein kleines Problem bleibt noch zu lösen. Betrügt der Kunde oder der Händler durch Kopieren des Geldscheins, dann kann die Bank zwar durch den obigen Mechanismus feststellen, dass der Betrug versucht wurde, sie kann jedoch nicht eindeutig nachweisen, *wer* den Betrug durchgeführt hat, i.e. soll die Bank das BKA zum Kunden – den sie ja gar nicht identifizieren kann – oder zum Händler schicken. David Chaum hat auch dieses Problem durch eine Verfeinerung des Protokolls gelöst. Angewendet wird ein Verfahren, das man *eingeschränkte blinde Signatur* nennt.

In der obigen Analogie funktioniert dies so, dass die Bank beide eingegangen Geldschein übereinanderlegt, gegen das Licht hält, und dann die Identität des Berügers feststellen kann. Ein Geldschein allein sagt nichts über die Identität des Betrügers aus. Diese Protokoll wird im Detail in Abschnitt [8.3] dargestellt.

8.2 Weitere grundlegende Protokolle

In diesem Abschnitt sehen wir uns drei grundlegende Protokolle an, die die Bausteine für die weiteren Protokolle zur Realisierung digitaler Münzen bilden. Dazu zählen **blinde Signaturen, Secret Splitting** und **Bit-Commitment** Protokolle.

8.2.1 Blinde Signaturen

Eine Signatur nennt man **Blinde Signatur**, wenn der Unterzeichner das Dokument nicht sieht. Für elektronisches Bargeld spielen blinde Signaturen eine wesentliche Rolle. Blinde Signaturen basieren auf folgender Idee, die auf David Chaum [39], [40] zurückgeht.

Will Bob ein Dokument M von Alice blind signieren lassen, dann verpackt er dieses Dokument, i.e. Bob verschlüsselt M. Er sendet das Chiffrat an Alice, die die verschlüsselte Nachricht signiert, durch die Verschlüsselung hat Alice keine Kenntnis über den Inhalt der Nachricht. Alice sendet das von ihr signierte Chiffrat anschließend zurück an Bob. Bob wiederum entpackt die Nachricht, i.e. entschlüsselt diese in solch einer Weise, dass die Signatur erhalten bleibt. Die Voraussetzung für das Funktionieren dieses Prozederes ist also, dass die Verschlüsselung und das Signieren vertauschbar sind, i.e. die zugrundeliegenden mathematischen Operationen müssen kommutieren.

Ein nicht-digitales Modell dieser blinden Signaturen besteht darin, dass Bob einen Brief zusammen mit einem Blatt Kohlepapier in einen Umschlag steckt. Bob läßt nun den 'Brief' von Alice unterschreiben, indem Alice ihre Unterschrift auf den Umschlag schreibt. Sie kennt also nicht den Inhalt des Briefes, durch das Kohlepapier wird ihre Unterschrift aber dennoch auf dem Brief erscheinen. Anschließend nimmt Bob den Brief wieder aus dem Umschlag und verfügt nun über einen von Alice blind unterschreibenen Brief.

In der Abbildung [8.6] sind die Schritte eines Verfahrens aufgeführt, das mit Hilfe des RSA-Verfahrens eine blinde Signatur realisiert, siehe auch das Ablaufdiagramm in Abbildung [8.7].

Wir verifizieren, dass s/k die signierte Nachricht ist. Es gilt:

$$\begin{aligned}\frac{s}{k} \bmod n &\equiv \frac{t^d}{k} \bmod n \\ &\equiv \frac{(k^e M)^d}{k} \bmod n \\ &\equiv \frac{k^{ed}}{k} M^d \bmod n \\ &\equiv M^d \bmod n,\end{aligned}$$

was genau die signierte Nachricht ist. Bei dieser Ableitung wird verwendet, dass

$$k^{ed} \equiv (k^e)^d \equiv k \bmod n$$

ist, was einfach die Verschlüsselung und anschließend die Entschlüsselung von k im RSA Algorithmus ist.

❶ **Szenario:**
Alice unterzeichnet blind eine von Bob erstellte Nachricht M.

❷ Alice wählt zwei zufällige große Primzahlen p, q und berechnet den RSA Modulus n.

❸ Alice wählt den Verschlüsselungsexponenten e und berechnet den Entschlüsselungsexponenten d.

❹ Alice veröffentlicht $K_{\text{pub}} = (e, n)$ und hält $K_{\text{priv}} = (d, n)$ geheim.

❺ Bob erstellt die Nachricht M.

❻ Bob besorgt sich den öffentlichen Schlüssel $K_{\text{pub}} = (e, n)$ von Alice.

❼ Bob wählt eine zufällige Zahl k, diese Zahl nennt man auch *Blendungsfaktor*, mit $\texttt{ggT}(k, n) = 1$ und berechnet

$$t \equiv k^e M \bmod n$$

❽ Bob sendet t an Alice.

❾ Alice signiert t, indem sie berechnet:

$$s \equiv t^d \bmod n.$$

Sie sendet s an Bob zurück.

❿ Bob berechnet

$$s/k \bmod n.$$

Dies ist die signierte Nachricht.

Abbildung 8.6: Ein Protokoll für blinde Signaturen, basierend auf dem RSA Algorithmus

Da die Zahl k zufällig gewählt ist, ist k^e mod n die RSA Verschlüsselung einer zufälligen Zahl und daher wieder zufällig. Daher gibt die Zahl $k^e M$ mod n keine Information über M[7]. Auf diese Art und Weise ist garantiert, dass Alice keinerlei Informationen aus der Nachricht gewinnen kann, die sie unterschreibt. Ist das Prozedere ausgeführt, verfügt Bob über die gleiche von Alice signierte Nachricht, als hätte Alice das Standardverfahren zur digitalen Signatur ausgeführt.

Beispiel:

Wir rechnen nun ein explizites Beispiel für die blinde Signatur durch. Wir verwenden die im Kapitel [6.3] im Beispiel gezeigte RSA Verschlüsselung mit

[7] Der Fall $M = 0$ ist hier nicht erlaubt.

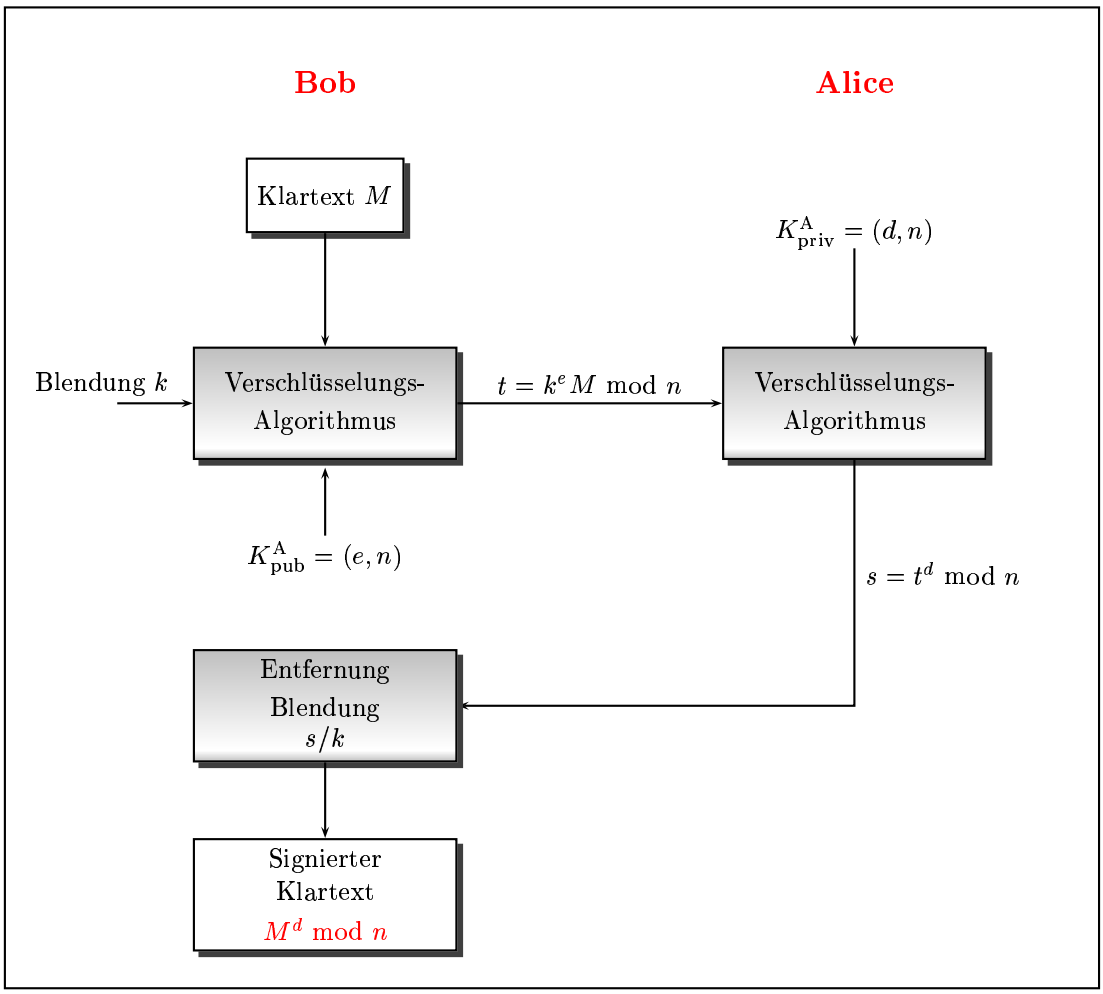


Abbildung 8.7: Ablauf des Protokolls der blinden Signatur mit dem RSA Verfahren

Parameter:

$$\begin{aligned} p = 7, \quad & q = 17, \\ n = 117, \quad & \phi(n) = 96, \\ e = 5 \quad & \Longrightarrow \; d = 77 \text{ mit } e \cdot d \equiv 1 \bmod \phi(n). \end{aligned}$$

① Das von Alice erzeugte Schlüsselpaar lautet mit diesen Parametern demnach:

$$K^A_{\mathrm{pub}} = (5, 119), K^A_{\mathrm{priv}} = (77, 119).$$

② Bob will nun den Buchstaben M mit ASCII$(M) = 77$ von Alice blind signieren lassen. Verwendet Alice das übliche Prozedere zur Erstellung

der digitalen Signatur mit dem RSA Verfahren (i.e. *ohne* Blendung), dann berechnet sie

$$77^{77} \bmod 119 \equiv 42 \bmod 119.$$

und Bob erhält den Wert[8] 42 als Signatur seines Textes.

③ Bob wählt nun einen Blendungsfaktor, i.e. eine Zahl k mit $0 < k < n$ und $\mathtt{ggT}(k, n) = 1$, zum Beispiel $k = 23$. Er berechnet:

$$\begin{aligned} t &= k^e M \bmod n \\ &= 23^5 \cdot 77 \bmod 119 \equiv 63 \bmod 119. \end{aligned}$$

Diese Zahl wird nun an Alice gesendet.

④ Alice signiert nun die Nachricht blind (i.e. sie kennt nicht den Wert von M), indem sie mit ihrem private Key (d, n) folgende Rechenoperationen ausführt:

$$\begin{aligned} s &= t^d \bmod n \\ &= 63^{77} \bmod 119 \\ &\equiv 14 \bmod 119. \end{aligned}$$

Der Wert s wird nun an Bob übertragen.

⑤ Bob entfernt nun den Blendungsfaktor indem er die folgende Rechnung ausführt:

$$\begin{aligned} \frac{s}{k} &\equiv s \cdot k^{-1} \bmod n \\ &\equiv 14 \cdot 88 \bmod 119 \\ &\equiv 42 \bmod 119. \end{aligned}$$

In dieser letzten Rechnung geht ein, dass

$$k \cdot k^{-1} \equiv 1 \bmod n$$

wobei man mit Hilfe des erweiterten EUKLIDischen Algorithmus den Wert $k^{-1} = 88$ bestimmt.

Fazit:

- Bob erhält die von Alice signierte Nachricht (Wert 42 mod 119)
- Alice hat keinerlei Kenntnis des Wertes von M, denn Bob hatte 77 gewählt, Alice sieht jedoch nur den geblendeten Wert 63 mod 119. Da der Blendungsfaktor k eine beliebige Zahl ist, die Bob geheim hält, hat Alice keine Möglichkeit, auf den Text zu schließen.

[8] Wie immer... die Antwort auf alles

8.2.2 Secret Splitting

Unter dem **Secret Splitting** versteht man das Zerteilen einer gegebenen Bitfolge M (das kann eine Nachricht, ein Dokument oder ein Schlüssel sein) in zwei (oder mehrere) Teile M_1, M_2. M_1 und M_2 allein sind wertlos, i.e. enthalten keinerlei Informationen über M. Zusammen erlauben sie jedoch die Rekonstruktion von M[9].

Hat der Text M die Länge von n Bit, dann erzeugt man eine n Bit Zufallszahl R und berechnet bzw. setzt

$$M_1 = M \oplus R$$
$$M_2 = R.$$

De facto stellt diese Form des Secret Splittings eine Verschlüsselung mit der One-Time-Pad Methode dar und ist daher absolut sicher. M_1 und M_2 enthalten keine Informationen über M, ist R eine echt zufällige Zahl, dann ist diese Aufteilung absolut sicher und es gilt:

$$M = M_1 \oplus M_2 = M \oplus R \oplus R.$$

Beispiel:

Sei

$$M = 0\,1\,1\,0 \quad 0\,1\,0\,0$$

und

$$R = 1\,1\,0\,0 \quad 1\,1\,1\,1$$

Dann ist

$$M_1 = 1\,0\,1\,0 \quad 1\,0\,1\,1$$
$$M_2 = 1\,1\,0\,0 \quad 1\,1\,1\,1.$$

Man erhält dann in der Tat:

$$M_1 \oplus M_2 = 0\,1\,1\,0 \quad 0\,1\,0\,0.$$

Dieses Verfahren läßt sich verallgemeinern, es findet in vielen Szenarien auch Anwendung. Beispielsweise soll ein Schlüssel für einen Bank-Tresor so unter fünf Angestellten aufgeteilt werden, dass der Tresor mit zwei beliebigen Teilschlüsseln geöffnet werden kann. Ein einzelner Teilschlüssel liefert keine Möglichkeit, den Tresor zu öffnen oder auf den Schlüssel zu schließen.

Allgemein definiert man das **(m,n)-Schwellenwertproblem** als die Aufgabe, ein Geheimnis in n Teile so aufzuteilen, dass es mit beliebigen m dieser Teile rekonstruiert werden kann, $m-1$ oder weniger Teile reichen nicht aus und liefern keinerlei Informationen über das Geheimnis.

[9] Secret Splitting wird unter anderem diskutiert in [215, Chap. 10], [78, Kap. 9.2] oder in SCHNEIERs Buch [193], Chapter 3.6.

8.2.3 Bit-Commitment Protokolle

Das grundlegende Ziel, was die sogenannten **Bit-Commitment Protokolle** realisieren ist folgendes Szeanrio. Angenommen, Alice soll sich auf einen bestimmten Wert eines Bits b festlegen, also ob $b = 0$ oder $b = 1$ ist. Sie möchte ihre Wahl jedoch geheim halten. Bob wiederum will eine Garantie dafür haben, dass Alice nach ihrer Festlegung des Bits diese Wahl nicht mehr ändert. Hierfür gibt es eine Reihe unterschiedlicher Protokolle[10], basierend auf symmetrischen Kryptosystemen, One-Way Funktionen oder Pseudozufallszahlen Generatoren.

Bit-Commitment mit symmetrischer Verschlüsselung

☞ **Commitment** (Festlegung)

1. Bob generiert eine zufällige Bitfolge R und sendet diese an Alice.
2. Alice generiert eine Nachricht, die aus dem Bit b besteht, auf das sie sich festlegt (das können auch mehrere Bits sein) und dem zufälligen Bitstring R von Bob. Alice verschlüsselt diese Nachricht mit einem symmetrischen Verschlüsselungssystem und sie wählt zu diesem Zweck einen zufälligen Schlüssel K. Das entstehende Chiffrat $E_K(R, b)$ sendet sie an Bob.

Diese beiden Schritte bilden den Teil der *Festlegung* (i.e. commitment) des Protokolls. Bob kann die Nachricht nicht entschlüsseln, da er nicht über den Key K verfügt. Daher weiß er auch nicht den Wert des Bits b.

Wenn es an der Zeit ist, dass Alice ihre Wahl offenlegen muß, geht das Protokoll mit folgenden Schritten weiter:

☞ **Überprüfung**

3. Alice sendet Bob den gewählten Schlüssel K.
4. Bob dechiffriert die Nachricht, um das Bit b offenzulegen. Er überprüft seinen zufälligen Bitstring R um die Gültigkeit des Bits zu verifizieren.

Falls die Nachricht nicht den zufälligen Bitstring von Bob enthält sondern nur aus dem Bit b besteht, könnte Alice das Chiffrat ohne Bobs Wissen mit unterschiedlichen Schlüsseln dechiffrieren bis sie einen Schlüssel findet, der ein anderes Bit produziert, als das, auf das sie sich festgelegt hat. Da das Bit nur zwei mögliche Werte hat, kann sie sicher sein, dass sie nach wenigen Versuchen erfolgreich ist. Die zufällige Bitfolge, die Bob erzeugt und die in die Festlegungphase eingeht, verhindert solch einen Angriff auf das Protokoll. Mit der zufälligen Bitfolge von Bob muß Alice eine neue Nachricht konstruieren, die nicht nur ihr Bit invertiert sondern auch die Zufallsfolge exakt reproduziert. Falls der Verschlüsselungsalgorithmus geeignet gewählt ist, ist die Wahrscheinlichkeit, dass ein solcher Betrugsversuch gelingt, vernachlässigbar.

Bit-Commitment mit One-Way Funktionen

[10] Siehe SCHNEIERs Buch [193], Kapitel 4.9.

☞ **Commitment** (Festlegung)

1. Alice legt sich auf den Wert des Bits b fest. Weiterhin generiert sie zwei zufällige Bitfolgen R_1 und R_2.
2. Alice wendet die öffentliche **One-Way Hash Funktion** H auf R_1, R_2 und b an.
3. Alice sendet den Hashwert und einen der zufälligen Bitfolgen $(H(R_1, R_2, b), R_1)$ an Bob.

Die Übertragung von Alice ist der Nachweis der Festlegung. Dadurch, dass Alice die Einwegfunktion H in Schritt 2 benutzt, um den Hashwert zu ermitteln, verhindert sie, dass Bob den Wert von b erfährt, denn die Einwegfunktion ist nicht invertierbar, so dass Bob den Wert von b erhalten kann.

☞ **Überprüfung**

1. Alice sendet Bob die Originalnachricht $[R_1, R_2, b]$.
2. Bob wendet die One-Way Hash Funktion auf die Nachricht an und vergleicht das Resultat und R_1 mit den Werten, die er in Schritt 3 erhalten hat. Stimmen diese Wert überein, ist der Wert von b korrekt.

Der Vorteil dieses Protokolls gegenüber dem ersten besteht darin, dass Bob keine Nachrichten an Alice senden muß. Alice sendet eine Nachricht an Bob, die ihre Festlegung enthält und eine zweite Nachricht für die Überprüfung. Ein zufälliger Bitstring von Bob ist nicht erforderlich, da das Resultat von Alices Festlegung eine Nachricht ist, die durch eine One-Way Funktion transformiert wird. Alice kann nicht betrügen und eine andere Nachricht $[R_1, R_2', b']$ finden mit

$$H[R_1, R_2', b'] = H[R_1, R_2, b].$$

Anmerkung:

Alice muß in diesem Protokoll zwei zufällige Bitfolgen R_1 und R_2 generieren.

☞ Die Folge R_1 sendet sie in Schritt 3 zusammen mit den Hash-Wert an Bob. Damit wird verhindert, dass Alice berügt, denn ohne R_1 kann sie leicht ein R_2 finden, so dass

$$H(R_2', \overline{b}) = H(R_2, b)$$

und Bob hat keine Möglichkeit zu prüfen, ob Alice ihre Wahl nicht doch geändert hat.

☞ Die Folge R_2 ist notwendig, damit Alice sicher sein kann, dass Bob den Wert von b nicht aufdecken kann. Ohne R_2 erhält Bob von Alice das Paar $H(R_1, b), R_1$. Da b nur zwei Werte annehmen kann, hat Bob in maximal zwei Durchläfen dn Wert von b berechnet.

8.3 Das Protokoll von D. Chaum, A. Fiat und M. Naor

Wir wollen nun die in Abschnitt [8.1] diskutierten Protokolle für elektronische Bargeld weiter verfeinern. Die Verwendung von Seriennummern und digitaler Signatur ermöglicht es der Bank, einen Betrugsversuch in Form von Mehrfachverwendung der E-Münze und eine Änderung des Wertes der Münze zu erkennen. Um die Anonymität des Kunden zu gewährleisten, wird die blinde Signatur der Münze durch die Bank durchgeführt. Dies kann durch das in Abschnitt [8.2.1] beschrieben Verfahren durchgeführt werden. Mit Hilfe der blinden Signatur wird einerseits die Anonymität gewährleistet, andererseits wird jeder Betrugsversuch erkannt, denn die Bank speichert die Seriennummer jeder eingegangenen Münze in einer Datenbank. Ist die Seriennummer einer eingehenden Münze bereits in der Liste enthalten, so kann darauf geschlossen werden, dass ein Betrugsversuch vorliegt. Der Betrüger bleibt jedoch anonym.

Protokoll 5:

In diesem Protokoll wird die Identität I von Alice auf jeder von ihr generierten Münze gespeichert. I besteht beispielsweise aus eine 30stelligen Kundennummer bei ihrer Bank. Allerdings wird die Identität mit einem **Secret Splitting Protokoll** unkenntlich gemacht.

1. Alice generiert n Münzen mit unterschiedlichen, zufälligen und langen Seriennummern wie im Protokoll 4. Nun spaltet sie ihre Identiätsnummer I mit einem Secret-Splitting Protokoll (siehe Abschnitt[8.2.2]) n mal unterschiedlich auf, was zu den n Identitätsfolgen

$$\begin{aligned} I_1 &= (I_{1L}, I_{1R}) \\ I_2 &= (I_{2L}, I_{2R}) \\ &\vdots \quad \vdots \\ I_n &= (I_{nL}, I_{nR}) \end{aligned}$$

führt.

2. In einem zweiten Schritt wendet Alice auf jede Hälfte jeder Identitätsfolge getrennt ein Bit-Commitment Protokoll an. Bei dem im Abschnitt [8.2.3] diskutierten Bit-Commitment Protokoll mit One-Way Hash Funktion hat jede der von Alice generierten Münze damit den folgenden Inhalt:

Betrag
Seriennummer
$[H(R_1, R_2, I_{1L}), R_1], [H(R_3, R_4, I_{1R}), R_3],$
$[H(R_5, R_6, I_{2L}), R_5], [H(R_7, R_8, I_{2R}), R_7],$
$\vdots$
$[H(R_{4n-3}, R_{4n-2}, I_{nL}), R_{4n-3}], [H(R_{4n-1}, R_{4n}, I_{nR}), R_{4n-1}].$

Dadurch ist jede Hälfte jeder Identitätsfolge separat unkenntlich gemacht und ist von keiner Partei veränderbar. Sie kann nur mit Alices Hilfe deco-

diert werden. Wegen der Verwendung eines Bit-Commitment Protokolls kann Alice bei der Decodierung nicht betrügen.

3. Alice verschlüsselt die so generierten n Münzen mit einem Protokoll für blinde Signaturen und schickt sie an die Bank.

4. Wie im Protokoll 4 verlangt die Bank nun von Alice die Entschlüsselung von $n-1$ zufällig ausgewählten Münzen. Die Bank überprüft Betrag und Seriennummer und fordert Alice auf, alle Identitätsfolgen offenzulegen.

5. Geht alles mit rechten Dingen zu, signiert die Bank die verbleibende Münze blind, belastet Alices Konto mit dem Betrag und gibt die signierte Münze an Alice zurück.

6. Alice dechiffriert ihre Münze (macht sie also wieder lesbar) und geht mit ihr Einkaufen.

7. Der Händler, der die Münze als Zahlungsmittel annimmt, überprüft die (blinde) Signatur der Bank.

8. Nun muß Alice von jeder der n Identitätsfolgen auf der Münze entweder die linke oder die rechte Hälfte öffnen. Der Händler legt eine zufällige Reihe der zu öffnenden Hälften fest.

9. Alice befolgt die Anweisung, i.e. gibt dem Händler

$$(R_1, R_2, I_{1L}), (R_5, R_6, I_{2L}), (R_{11}, R_{12}, I_{3R}), \ldots . \tag{8.1}$$

10. Der Händler bringt die Münze zur Bank mit der Bitte um Gutschrift auf sein Konto.

11. Die Bank überprüft ihre (eigene) Signatur. Weiterhin überprüft sie in ihrer Datenbank, ob eine Münze mit der gleichen Seriennummer bereits eingegangen ist. Ist dies nicht der Fall, erhält der Händler den Wert der Münze gutgeschrieben und die Bank protokolliert die Seriennummer der Münze mit allen Identitätsmerkmalen in ihrer Datenbank.

12. Ist die Seriennummer bereits in der Datenbank vorhanden, dann liegt mit großer Wahrscheinlichkeit ein Betrugsversuch vor. Nun werden die verschlüsselten Identitätsdaten auf der Münze mit denen in der Datenbank verglichen. Stimmen sie überein, dann weiß die Bank, dass der Händler die Münze kopiert hat.

    Im anderen Fall ist der ursprüngliche Eigentümer der Münze, nämlich Alice, der Betrüger. Sie hat offenbar mit zwei Kopien der Münzen bei einem Händler A und bei einem anderen Händler B bezahlt. Nun vergleicht die Bank alle Identitätspaare auf der Münze mit den gespeicherten in der Datenbank (zu der doppelten Seriennummer). Da der Händler A in Schritt 8 sicherlich nicht exakt die gleichen n Hälften der Identitätsfolgen zu Offenlegung ausgewählt hat wie Händler B, gibt es mindestens ein Paar, bei dem beide Hälften geöffnet sind. Die Bank verknüpft diese beiden Hälften (z.B. mit `XOR`) und erhält damit die Identität von Alice im Klartext.

Alice und der Händler haben bei diesem Protokoll keine Möglichkeit zum Betrug. Wie bei klassischem Bargeld hat aufgrund der Anonymität auch hier ein Dieb die Möglichkeit des Betrugs. Das hier vorgestellte Protokoll zeigt schon eine gewisse Komplexität auf. Es ist alles andere als trivial, die Korrektheit und Sicherheit dieses Protokolls zu verifizieren. Dies ist in den Arbeiten von CHAUM [41, 43] ausgeführt.

8.4 Das Protokoll von Stefan Brands

☞ **Teilnehmer**
Die Teilnehmer des Verfahrens sind drei Parteien: Die Bank, der Ausgeber Alice, i.e. die Person, die mit der elektronischen Münze bezahlt und der Händler.

☞ **Initialisierung**
Die Initialisierung des Procederes wird einmal von einer zentralen, autorisierten Stelle ausgeführt.

a) Wähle eine große Primzahl p, so dass

$$q = (p-1)/2$$

ebenfalls Primzahl ist[11].

b) Sei g das Quadrat einer primitiven Wurzel modulo p. Dies gewährleistet die Äquivalenz:

$$g^{k_1} \equiv g^{k_2} \bmod p \iff k_1 \equiv k_2 \bmod q.$$

c) Wähle zwei zufällige Exponenten l_1 und l_2. Berechne

$$g_1 = g^{l_1} \bmod p; g_2 = g^{l_2} \bmod p.$$

Die beiden Exponenten l_1, l_2 werden verworfen.

d) Die drei Zahlen g, g_1, g_2 werden veröffentlicht.

e) Zwei öffentliche Hash-Funktionen H und H_0 werden gewählt. H hat als Input fünf ganze Zahlen und als Output eine ganze Zahl modulo q. H_0 hat vier ganze Zahlen als Input und produziert ebenfalls eine ganze Zahl modulo q.

☞ **Bank**
Die Bank wählt eine geheime Nummer x und berechnet

$$h \equiv g^x \bmod p; h_1 \equiv g_1^x \bmod p; h_2 \equiv g_2^x \bmod p$$

Die drei Zahlen h, h_1, h_2 werden veröffentlicht und identifizieren die Bank.

[11] Das Verfahren, solch eine Zahl zu finden, besteht darin, eine zufällige Primzahl q zu wählen und anschließend testet man beispielsweise mit dem RABIN-MILLER Verfahren, ob die Zahl $2q+1$ eine Primzahl ist.

☞ **Alice**
Der Teilnehmer, der die digitale Münze verwenden – und ausgeben – will, wählt eine geheime Identitätsnummer u und berechnet die Kontonummer

$$I \equiv g_1^u \bmod p.$$

Diese Nummer I wird an die Bank gesendet, die wiederum I mit Informationen speichert, die Alice identifiziert (e.g. ihr Name, Adresse, Phone-Nummer). Alice sendet *nicht* die Zahl u. Die Bank sendet

$$z' \equiv (Ig_2)^x \bmod p$$

an Alice zurück.

☞ **Der Händler**
Der Händler wählt irgend eine Identifikationsnummer M und läßt sich darüber bei der Bank registrieren.

☞ **Erzeugung einer Münze**
Um eine neue Münze zu generieren, kontaktiert Alice die Bank und bittet um eine Münze. Die Bank fordert einen Nachweis der Identität, dies ist genau wie die klassische Banktransaktion, wenn jemand von seinem Konto Geld abhebt. In dem aktuellen Verfahren haben alle Münzen den gleichen Wert. Eine Münze wird durch ein 6-Tupel von Zahlen dargestellt:

$$(A, B, z, a, b, r).$$

Die Komplexität dieser elektronischen Münzen ist notwendig, um einerseits Anonymität zu gewährleisten und andererseits zu verhindern, dass eine Münze zweimal ausgegeben wird.

Die Erzeugung einer elektronischen Münze geschieht nach folgendem Verfahren:

1. Die Bank wählt eine zufällige Zahl w (für jede Münze wird ein anderer Wert gewählt), und berechnet

   $$g_w = g^w \bmod p$$

   und

   $$\beta = (Ig_2)^w \bmod p.$$

   Die Werte (g_w, β) werden an Alice übertragen.
2. Alice wählt fünf zufällige Zahlen, die sie geheim hält:

   $$(s, x_1, x_2, \alpha_1, \alpha_2)$$

3. Alice berechnet nun folgende Größen:

   $$\begin{aligned} A &= (Ig_2)^s \bmod p \\ B &= g_1^{x_1} g_2^{x_2} \bmod p \\ z &= z^{'s} \bmod p \\ a &= g_w^{\alpha_1} g^{\alpha_2} \\ b &= \beta^{s\alpha_1} A^{\alpha_2} \bmod p \end{aligned}$$

Münzen mit $A = 1$ sind nicht zulässig, was in zwei Fällen auftreten kann. Der eine Fall tritt dann ein, wenn $s \equiv 0 \bmod q$, daher fordert man $s \not\equiv 0$. Der andere Fall tritt dann ein, wenn $Ig_2 \equiv 1 \bmod p$. In diesem Fall ist es Alice gelungen, durch eine glückliche Wahl von u das diskrete Logarithmusproblem zu lösen. Daher muß die Primzahl p groß genug gewählt werden, dass dieser Fall realistisch nicht eintreten kann.

4. Alice berechnet

$$c \equiv \alpha_1^{-1} H(A, B, z, a, b) \bmod q,$$

und sendet c an die Bank. Hier ist H die öffentliche Hash-Funktion, die oben erwähnt wurde.

5. Die Bank berechnet

$$c_1 \equiv cx + w \bmod q$$

und sendet c_1 an Alice zurück.

6. Alice berechnet

$$r \equiv \alpha_1 c_1 + \alpha_2 \bmod q.$$

Die Münze (A, B, z, a, b, r) ist nun vollständig konstruiert. Der Betrag der Münze wird von Alices Bankkonto abgebucht.

Das Verfahren, das in der Praxis recht schnell ist, wird für jede zu erstellende Münze durchgeführt. Für jede Transaktion sollte die Bank eine neue zufällige Zahl w erzeugen, ähnlich sollte Alice für jede Münze ein neues zufälliges 5-Tupel $(s, x_1, x_2, \alpha_1, \alpha_2)$ wählen.

☞ **Eine Münze ausgeben**
Alice übergibt die Münze (A, B, z, a, b, r) an den Händler. Dabei wird folgendes Verfahren durchlaufen:

1. Der Händler prüft:

$$g^r \equiv a h^{H(A,B,z,a,b)} \bmod p,$$
$$A^r \equiv z^{H(A,B,z,a,b)} b \bmod p.$$

Falls die Werte übereinstimmen, dann weiß der Händler, dass die Münze gültig ist. Es sind jedoch weitere Schritte notwendig um zu überprüfen, dass keine Mehrfachvergabe der Münze vorliegt.

2. Der Händler berechnet

$$d = H_0(A, B, M, t),$$

wobei H_0 die zweite öffentliche Hash-Funktion mit vier Inputwerten ist, die in der Initialisierungsphase festgelegt wurde. t ist eine Zahl, die Datum und Uhrzeit der Transaktion repräsentiert. Diese Zahl t wird an dieser Stelle eingebaut, damit verschiedene Transaktionen auch unterschiedliche d-Werte haben. Der Händler sendet d an Alice.

3. Alice berechnet

$$r_1 \equiv dus + x_1 \bmod q$$

und

$$r_2 \equiv ds + x_2 \bmod q$$

wobei u die Geheimnummer von Alice ist und s, x_1, x_2 Teil des geheimen zufälligen 5-Tupels ist, das Alice früher gewählt hat. Alice sendet r_1 und r_2 an den Händler.

4. Der Händler prüft, ob gilt

$$g_1^{r_1} g_2^{r_2} \equiv A^d B \bmod p.$$

Falls diese Kongruenz gilt, dann akzeptiert der Händler die Münze, andernfalls verweigert er die Annahme.

☞ **Der Händler hintelegt die Münze in der Bank**
Einige Tage nachdem der Händler die Münze von Alice erhalten hat, will er diese auf seine Bank hinterlegen. Dazu überträgt er die Münze (A, B, z, a, b) zusammen mit dem Tripel (r_1, r_2, d) an die Bank. Die Bank führt daraufhin folgende Schritte durch:

1. Die Bank überprüft, dass die Münze $(A, B, z, a, b,)$ zuvor noch nicht bei ihr hinterlegt wurde. Wenn dies der Fall ist, geht sie zum nächsten Schritt über. Falls die Münze bereits hinterlegt wurde, springt die Bank zu einem der Verfahren zur Betrugskontrolle, die unten diskutiert werden.
2. Die Bank prüft, ob die folgenden Kongruenzen gelten:

$$\begin{aligned} g^r &\equiv a h^{H(A,B,z,a,b)} \bmod p \\ A^r &\equiv z^{H(A,B,z,a,b)} b \bmod p \\ g_1^{r_1} g_2^{r_2} &\equiv A^d B \bmod p. \end{aligned}$$

Wenn dies der Fall ist, ist die Münze gültig und der Wert wird auf das Konto des Händlers gutgeschrieben.

☞ **Betrugs-Kontrolle**
Es gibt eine Reihe von Möglichkeiten in diesem Verfahren zu betrügen. Hier sind verschiedene Verfahren, diesem entgegen zuwirken.

1. Alice gibt die Münze zwei Mal aus. Das erste Mal verfährt sie wie oben mit dem Händler, das zweite Mal mit jemanden, den wir Verkäufer nennen.

Kapitel 9

Weitere Protokolle

9.1 The Dining Cryptographers Problem

In [40] wird von DAVID CHAUM das *Dining Cryptographers Problem* beschrieben, das ein Verfahren zur **Senderanonymität** darstellt.

> Drei Kryptographen sitzen an einem Tisch in ihrem bevorzugten 3-Sterne Restaurant. Der Inhaber des Restaurants informiert sie nach dem Essen, dass ein Arrangement getroffen wurde, das Essen anonym zu zahlen. Das Arrangement sieht so aus, dass entweder einer der Kryptographen oder die NSA die Rechnung begleicht. Jeder der drei respektiert die Anonymität des bezahlenden Kryptographen, wollen aber sicher sein, wenn das NSA bezahlt.

Wie gelingt es den drei Kryptographen, die Alice, Bob und Carol heißen, festzustellen, ob einer der drei gezahlt hat und dennoch dessen Anonymität zu gewährleisten?

Dieses Problem wurde von CHAUM folgendermaßen gelöst. Jeder Kryptograph wirft eine faire Münze hinter seiner Speisekarte, so dass nur der rechts neben ihm sitzende Kryptograph und er selbst das Ergebnis sehen kann. Jeder Kryptograph sagt nun für alle hörbar, ob die beiden Münzen, die er sehen konnte – diejenige, die der links neben ihm sitzende und die, die er selbst geworfen hat – auf die gleiche Seite gefallen sind oder auf unterschiedliche Seiten. Ist einer der Kryptographen derjenige, der die Rechnung bezahlt, sagt er das Gegenteil dessen, was er sieht. Wird eine ungerade Anzahl von unterschiedlichen Ausgängen am Tisch geäußert, ist dies ein Indiz dafür, dass einer der Kryptographen bezahlt hat, eine gerade Anzahl von Unterschieden sagt aus, dass die NSA bezahlt hat[1]. Falls dennoch einer der Kryptographen bezahlt, weiß keiner der anderen, wer es war, i.e. der Bezahler bleibt anonym.

Um zu sehen, dass dies funktioniert, stellen wir uns vor, Alice will herausfinden, welcher der anderen Kryptographen bezahlt hat, wobei wir annehmen, dass weder sie noch die NSA bezahlt hat. Falls sie zwei verschiedene Münzseiten sieht, dann sagten die beiden anderen Kryptographen, Bob und Carol, entweder glei-

[1] Vorausgesetzt, es wird nur einmal bezahlt.

che Ausgänge oder ungleiche Ausgänge (da im Fall, dass einer der Kryptographen bezahlt hat, eine ungerade Anzahl von 'ungleich' geäußert werden). Falls beide 'ungleich' sagen, dann ist derjenige, der bezahlt hat, der Kryptograph, der am nächsten zu der Münze sitzt, die gleich der versteckten Münze ist (das ist die, die Bob und Carol sehen können). Falls beide 'gleich' sagen, ist der Bezahler der Kryptograph, der am nähesten zur der Münze sitzt, die verschieden von der Münze ist, die Alice nicht sehen kann.

Falls Alice zwei Münzen sieht, die gleich sind, dann hat entweder Bob 'gleich' gesagt und Carol 'ungleich' oder umgekehrt, Bob hat 'ungleich' gesagt und Carol 'gleich'. Ist die versteckte Münze die gleiche, wie die beiden, die sie sieht, dann ist der Kryptograph, der 'ungleich' sagt derjenige, der bezahlt hat. Ist die versteckte Münze unterschiedlich zu den beiden, die sie sieht, dann ist der Kryptograph, der 'gleich' sagt derjenige, der bezahlt hat. In allen Fällen muß Alice den Ausgang des Münzwurfs zwischen Bob und Carol kennen, um festzustellen, wer das Essen bezahlt hat.

9.2 Oblivious Transfer

Kapitel 10

Authentifikation und Hash Funktionen

Unter dem Begriff **Authentifikation** versteht man einen Prozess, der bestimmte Informationen verifiziert und/oder den Ursprung und die Integrität der Information beweist. Szenarien, die Authentifikation einsetzen, sind beispielsweise:

- ☞ die Verifikation des Ursprungs eines Dokumentes
- ☞ die Verifikation der Identität eines Senders
- ☞ die Verifikation der Uhrzeit und des Datums, wann ein Dokument erstellt oder abgesendet wurde
- ☞ die Identität eines Computers oder eines Benutzers, letzters beispielsweise durch den Login Prozess

Eine **digitale Signatur** ist ein kryptographisches Verfahren, mit dessen Hilfe diese Szenarien umgesetzt werden. Die digitale Signatur eines Dokumentes ist eine bestimmte Information, die sowohl von dem zu authentifizierenden Dokument als auch vom Private Key des Senders abhängt. Die digitale Signatur verwendet typischerweise eine sogenannten **Hash Funktion** und die Verschlüsselung mit dem Private Key des Senders. Es gibt jedoch auch andere Modelle, die die Authentifikation realisieren.

Tagtäglich unterschreiben Personen Briefe, Kreditkartenbelege und andere Dokumente mit ihrem Namen. Damit wird dokumentiert, dass die unterschreibende Person mit dem Inhalt des Dokumentes einverstanden ist. Mit anderen Worten, die unterschreibenden Personen authentifizieren sich als Absender oder Urheber des Dokumentes. Dieses Verfahren ermöglicht es anderen Personen nachzuvollziehen, dass eine bestimmte Nachricht tatsächlich von demjenigen stammt, der unterschrieben hat. Dieses Verfahren ist natürlich nicht narrensicher, denn Unterschriften können gefälscht werden oder der Inhalt eines Dokumentes kann nach der Unterschrift geändert werden.

Die Verfahren der digitalen Signatur und der handschriftliche Unterschriften basieren beide auf der Tatsache, daß es sehr schwierig ist, zwei Personen mit der

gleichen Signatur zu finden. Digitale Signaturen werden mit Hilfe von Public-Key Kryptosysteme erstellt, indem ein eindeutiges Charakteristikum einer Person zugeordnet wird. Werden Public-Key Verfahren zur *Verschlüsselung* eingesetzt, chiffriert der Sender die Nachricht mit dem Public-Key des beabsichtigten Empfängers. Werden Public-Key Verfahren eingesetzt, um digitale Signaturen zu berechnen, verschlüsselt der Sender den 'digitalen Fingerabdruck' der Nachricht mit seinem Private-Key. Jeder, der Zugriff auf den Public-Key des Empfängers hat, kann die Signatur verifizieren.

Angenommen, Alice möchte ein signiertes Dokument oder eine signierte Nachricht an Bob senden. Allgemein besteht der erste Schritt darin, eine Hash-Funktion auf die Nachricht anzuwenden. Damit wird ein sogenannter **Message Digest** generiert. Der Message Digest ist üblicherweise beträchtlich kleiner als die ursprüngliche Nachricht. Die Aufgabe der Hash-Funktion besteht darin, eine Nachricht beliebiger Länge auf einen Wert fester Länge abzubilden. Um eine digitale Signatur aus dem Message Digest zu erzeugen wird lediglich der Message Digest verschlüsselt und nicht die komplette Nachricht. Diese Vorgehensweise hat den Vorteil, wesentlich effektiver zu sein, da die Verschlüsselung des kurzen Message Digests wesentlich schneller möglich ist als die eines kompletten Dokumentes. Andererseits erzeugt dieses Verfahren eine Unsicherheit.

Alice sendet an Bob den verschlüsselten Message Digest und die Nachricht, die sie verschlüsselt hat oder auch nicht. Damit Bob die Echtheit der Signatur überprüfen kann, wendet Bob die gleiche Hash Funktion wie Alice auf die erhaltene Nachricht (im Klartext) an. Damit erhält Bob ebenfalls einen Message Digest. Dann dechiffriert er den ebenfalls erhaltenen, verschlüsselten Message Digest von Alice und vergleicht beide Werte. Falls die Werte übereinstimmen, hat Bob erfolgreich die Echtheit der Signatur bewiesen. Falls die beiden nicht übereinstimmen, können eine Reihe von Gründen vorliegen:

- ☞ irgend jemand versucht sich als Alice auszugeben
- ☞ die Nachricht wurde – seit Alice sie signiert hat – verändert
- ☞ ein Übertragungsfehler ist aufgetreten

Wie oben angedeutet, hat diese Form der digitalen Signatur eine prinzipielle, konstruktive Unsicherheit. Es darf nicht der Fall eintreten, dass zwei verschiedene Nachrichten durch die Hash Funktion auf den gleichen Message Digest abgebildet werden. Werden zwei Nachrichten auf den gleichen Hashwert abgebildet, ist eine sogenannte **Kollision** aufgetreten. Eine notwendige Sicherheitsanforderung für die meisten digitalen Signatur Schema sind **kollisionsfreie Hash Funktionen**. Eine Hash Funktion gilt als sicher, falls es sehr zeitaufwändig ist – wenn es überhaupt möglich ist – aus dem gegebenen Message Digest auf den ursprünglichen Text zu schließen. Es existiert jedoch eine Angriffsart, genannt **Geburtstagsattacke**, dass es einfacher ist, zwei Nachrichten zu finden, die auf den gleichen Hash Wert abgebildet werden als eine Nachricht zu finden, die auf den vorgebenen Hashwert abgebildet wird. Der Name dieser Angriffsart stammt aus der Wahrscheinlichkeitsrechnung aus dem Problem, die Wahrscheinlichkeit dafür zu bestimmen, dass zwei oder mehr Personen am gleichen Tag Geburtstag haben.

Eine weitere Schwachstelle dieser digitalen Signatur mit Public-Key Verfahrens liegt darin, daß irgendjemand vorgeben kann, er/sei sei Alice und signiert mit einem Schlüsselpaar, das angeblich Alice gehört. Um solche Szenarios zu vermeiden, gibt es digitale Dokumente, die man **Zertifikate** nennt. Die Aufgabe der Zertifikate ist der Nachweis der eindeutigen Zuordnung eines öffentlichen Schlüssels zu einer bestimmten Person (oder einer bestimmten Maschine).

Digital Zeitstempel können im Zusammenhang mit digitalen Signaturen dazu verwendet werden, den Zeitpunkt der Erstellung eines Dokumentes an das Dokument zu binden. Das Datum und die Uhrzeit einfach in der Nachricht einzugeben ist natürlich nicht ausreichend, da Datum und Uhrzeit auf Rechnern problemlos manipuliert werden können.

10.1 Ziele der Authentifikation

Im Rahmen der Netzwerkkommunikation können eine Reihe von Angriffsmethoden identifiziert werden, die in folgende Kategorien eingeteilt werden können (siehe [194] für eine umfassende Diskussion dieser Angriffsmethoden):

- **Offenlegung** (Disclosure)
  Darunter versteht man die Offenlegung des Inhalts einer Nachricht an irgendeine Person (oder einen Prozess), der nicht über den geeigneten Schlüssel verfügt.

- **Traffic Analysis oder Verkehrsflussanalyse**
  Diese Angriffsart beinhaltet die Analyse des Musters des Datenverkehrs zwischen zwei Teilnehmern. In einer verbindungsorientierten Anwendung kann damit die Häufigkeit und die Dauer von Verbindungen und Übertragungen bestimmt werden. In verbindungslosen und verbindungsorientierten Übertragungen kann die Länge von Nachrichten und die Anzahl der ausgetauschten Nachrichten ermittelt werden.

- **Maskierung**
  Unter Maskierung versteht man die Einschleusung von Nachrichten durch eine betrügerische Quelle. Diese Angriffsart beinhaltet auch die Einschleusung von Nachrichten durch einen Angreifer. Die Intention dabei ist, dem Empfänger die eingeschleuste Nachricht als Nachricht eines autorisierten Benutzers auszugeben. Umgekehrt beinhaltet diese Angriffsart auch die betrügerische Bestätigung – oder Nicht-Bestätigung durch nicht autorisierte Personen.

- **Modifikation einer Nachricht**
  Diese Angriffsart bedeutet die nicht autorisierte Änderung des Inhaltes einer Nachricht, beispielsweise durch Einfügen, Löschen oder Umschreiben.

- **Modifikation der Reihenfolge**
  Any modification to a sequence of messages between parties, including insertion, deletion and reordering.

- **Timing modification**
  Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g. datagram) could be delayed or replayed.

- **Repudiation**
  This means the denial of the receipt of a message by destination or the other way, the denial of transmission of message by source.

Gegenmaßnahmen zur Abwendung der ersten beiden Angriffsarten sind typischerweise symmetrische Verschlüsseliungssysteme. Um die zweite bis sechste Art abzublocken, wird die Nachrichtenauthenifizierung eingesetzt. Die Gegenmaßnahme um ein System gegen die letzte Angriffsart resistent zu machen ist

die digitale Signatur. Digitale Signaturverfahren blocken eine Reihe anderer Verfahren ebenfalls ab.

Zusammenfassend kann die Authenifikation und digitale Signatur folgendermaßen definiert werden:

> Die Message Authentifikation ist ein Verfahren mit dessen Hilfe verifiziert werden kann, dass eine empfangene Nachricht von dem angegebenen Sender stammt und während der Übertragung nicht verändert wurde. Die Message Authentifikation kann auch die Reihenfolge von Nachrichten und der Zeitpunkt der Übertragung verifizieren. Die digitale Signatur ist ein Authentifikationsverfahren zur Gewährleistung der Verbindlichkeit sowohl des Nachrichtensenders als auch -empfängers.

10.2 Möglichkeiten zur Authentifikation

Prinzipiell besteht jeder Mechanismus zur Authentifikation einer Nachricht aus zwei Stufen:

1. Auf der unteren Ebene benötigt man eine bestimmte Funktion, die einen **Authentikator** generiert, Dies ist beispielsweise ein numerischer Wert, der dazu verwendet wird, eine Nachricht *eindeutig* zu authentifizieren.
2. Der Authentikator wird dann als Grundbaustein eines höheren Level Protokolls verwendet, welches den Empfänger der Nachricht ermöglicht, die Authentizität einer Nachricht zu verifizieren.

Die Funktionen – i.e. die verschiedenen Möglichkeiten – die eingesetzt werden können, um auf der unteren Ebene einen Authentikator zu erzeugen, können in drei Kategorien eingeteilt werden:

- ✕ **Nachrichtenverschlüsselung**
  Bei dieser Form der Authentifikation dient der Chiffretext der kompletten Nachricht als Authentikator.
- ✕ **Message Authentication Code – MAC**
  Bei der Authentifikation mittels Message Authentication Code wird eine öffentlich bekannte Funktion auf die Nachricht angewendet. Dadurch wird einen Wert mit fester Länge generiert. Dieser Wert dient als Authentikator der Nachricht. Mit einem geheimen Schlüssel wird entweder zuerst die Nachricht oder anschließend der Authentikator chiffriert.
- ✕ **Hash Funkionen**
  Eine öffentlich bekannte Funktion bildet eine Nachricht beliebiger Länge auf einen festen Zahlenwert ab. Diesen Zahlenwert nennt man **Hash-Wert**. Dieser Hash-Wert dient als Authentikator.

Im folgenden sehen wir uns diese drei Verfahren explizit an.

10.2.1 Nachrichtenverschlüsselung

Die Verschlüsselung von Nachrichten kann ein gewisses Mass an Authentifikation bereitstellen. Die Betrachtung ist für symmetrische und asymmetrische Verschlüsselung unterschiedlich.

Konventionelle, symmetrische Verschlüsselung

Wir betrachten die übliche Verschlüsselung mit einem symmetrischen Kryptosystem, siehe Abbildung [10.1]. Eine verschlüsselte Nachricht wird vom Sender Alice zum Empfänger Bob übertragen. Diese Nachricht ist mit einem geheimen Schlüssel K chiffriert, dieser Schlüssel ist nur Alice und Bob bekannt. Falls kein Dritter Kenntnis dieses Schlüssels hat, ist die Eigenschaft *Vertraulichkeit* gewährleistet: Niemand ausser Alice und Bob kann den Klartext aus der Nachricht rekonstruieren.

Weiterhin kann festgestellt werden, dass Bob sicher sein kann, dass die erhaltene Nachricht von Alice generiert wurde. Warum ist das der Fall? Die Nachricht

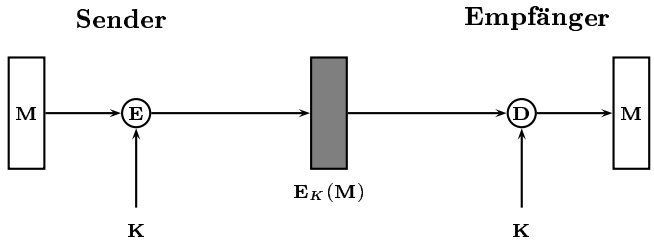


Abbildung 10.1: Nachrichtenverschlüsselung: Symmetrische Verschlüsselung

muss von Alice stammen, da Alice die einzige andere Partei ist, die im Besitz des Schlüssels K ist. Daher ist Alice auch der einzige Teilnehmer, dem die notwendige Information zur Verfügung steht, um den Chiffretext zu erzeugen, der mit dem Schlüssel K dechiffriert werden kann. Zusätzlich kann Bob sicher sein, daß keines der Bits des Klartext während der Übertragung verändert wurde, falls der Klartext aus dem erhaltenen Chiffrat rekonstruiert werden kann. Denn ein Angreifer, der den geheimen Schlüssel K nicht kennt, hat keine Möglichkeit, über Änderung der Bits des Chiffretexts den Klartext gezielt zu manipulieren.

Bob kann daher mit dieser Methode verifizieren, dass Alice die Urheberin der Nachricht ist. Da aber sowohl Alice als auch Bob den Schlüssel K kennen, ist dies nicht ausreichend, Dritten gegenüber zu beweisen, dass Alice eindeutig Urheberin einer Nachricht ist.

Public-Key Verschlüsselung

Die direkte Verwendung der Verschlüsselung mit einem Public-Key Verfahren wie in der Abbildung [10.2] dargestellt, liefert Vertraulichkeit aber keine Authentifikation. Der Sender – Alice – benutzt Bobs öffentlichen Schlüssel K^B_{pub} zur Verschlüsselung der Nachricht. Da nur Bob den passenden Private Key K^B_{priv} zur Verfügung hat, kann nur er die Nachricht dechiffrieren. Dieses Verfahren gewährleistet natürlich keine Authentifikation, da jeder Angreifer Bobs Public Key verwenden kann, um unter der Identität Alice eine verschlüsselte Nachricht an Bob zu senden.

Um eine Authentifikation mit Public-Key Kryptosystemen zu realisiseren, benutzt Alice ihren privaten Schlüssel K^A_{priv} um die Nachricht zu verschlüsseln. Bob, der Empfänger der Nachricht, der sich davon überzeugen will, dass Alice der Absender ist, besorgt sich deren Public-Key K^A_{pub} und dechiffriert den erhaltenen Chiffretext, siehe Abbildung [10.3]. Dieses Verfahren liefert ein Maß an Authentifikation wie im symmetrischen Fall: Die Nachricht muß von Alice stammen, da Alice die einzige Person ist, die den privaten Schlüssel K^A_{priv} besitzt. Daher ist sie die einzige Person, die einen Chiffretext erzeugen kann, der mit K^A_{pub} wieder dechiffriert werden kann.

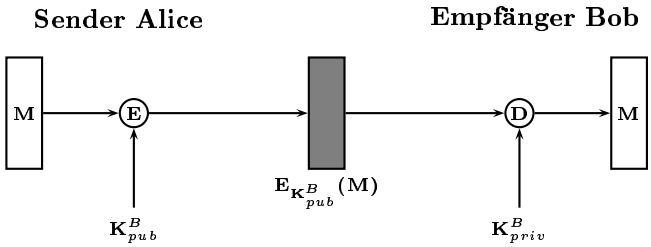


Abbildung 10.2: Verschlüsselung einer Nachricht: Vertraulichkeit mit Public-Key Kryptosystemen.

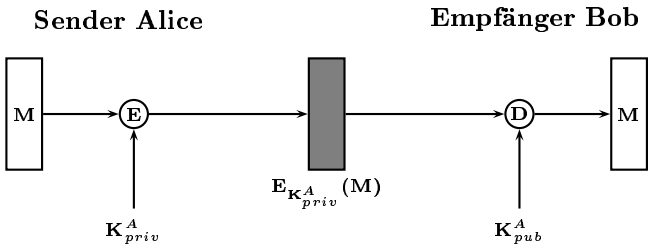


Abbildung 10.3: Verschlüsselung einer Nachricht: Authentifikation und Signatur mit Public-Key Kryptosystemen.

Das in der Abbildung [10.3] dargestellte Schema liefert daher Authentifizierung. Das Verfahren gewährleistet aber auch die **digitale Signatur**[1] Nur Alice kann den Chiffretext konstruiert haben, da ausschließlich Alice im Besitz von K^A_{priv} ist. Selbst Bob – der Empfänger der Nachricht – kann den Chiffretext nicht erzeugt haben. Falls daher Bob im Besitz des Chiffretexts ist, hat Bob den eindeutigen Beweis zur Verfügung – auch gegenüber Dritten – daß die Nachricht von Alice stammt. Dieser Beweis hat auch einem Dritten gegenüber Gültigkeit. Mit anderen Worten, Alice hat die Nachricht 'unterschrieben', indem sie die Message mit ihrem Private Key verschlüsselt hat.

Dieses Schema gewährleistet keine Vertraulichkeit, denn jeder, der im Besitz von Alice's Public-Key ist, kann den Chiffretext entschlüsseln.

Um beide Anforderungen, also sowohl Vertraulichkeit als auch Authentifikati-

[1] In praktischen Anwendungen wird die digitale Signatur in einer anderen Art und Weise implementiert. Im Prinzip funktioniert das Verfahren jedoch exakt auf diese Weise.

on zu gewährleisten, verschlüsselt Alice zuerst den Klartext mit ihrem Private Key; dieser Schritt gewährleistet die digitale Signatur. Anschließend benutzt Alice den Public Key und verschlüsselt damit ein zweites Mal. Dieser Schritt gewährleistet die Vertraulichkeit. Dieses Verfahren ist in der Abbildung [10.4] dargestellt. Der Nachteil dieser Methode liegt darin, dass Public-Key Algorithmen, die in der Regel sehr komplex sind, viermal angewendet werden müssen. Dies ist ein sehr zeitaufwändiges Prozedere.

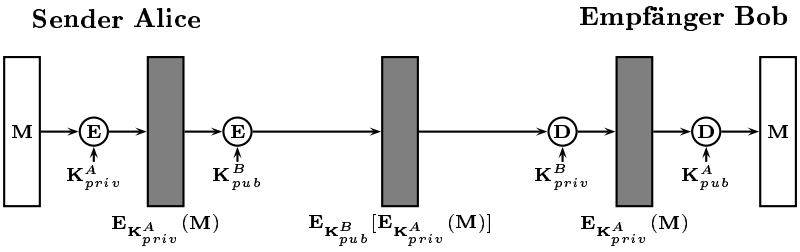


Abbildung 10.4: Nachrichtenverschlüsselung: Public-key Verfahren: Vertraulichkeit, Authentifikation und Signatur.

In der Tabelle [10.1] sind die verschiedenen Aspekte dieser Verfahren hinsichtlich Vertraulichkeit und Authentifikation zusammengefaßt.

Symmetrische Verschlüsselung

Alice $\longrightarrow$ Bob : $\mathrm{E}_{\mathrm{K}}[M]$

- Gewährleistet Vertraulichkeit
  Nur Alice und Bob kennen den Schlüssel K.
- Gewährleistet einen gewissen Grad an Authentifikation
  - Aus Bobs Sicht kann die Nachricht nur von Alice stammen
  - Nachricht wurde während der Übertragung nicht modifiziert
- Gewährleist keine digitale Signatur
  - Der Empfänger könnte die Nachricht fälschen
  - Der Sender könnte das Absenden der Nachricht abstreiten

Asymmetrische Verschlüsselung

Alice $\longrightarrow$ Bob : $\mathrm{E}_{\mathrm{K}^B_{pub}}[M]$

- Gewährleistet Vertraulichkeit
  - Ausschließlich Bob besitzt den Schlüssel K^B_{priv} zum Dechiffrieren
- Gewährleistet keine Authentifikation
  - Jeder Kommunikationsteilnehmer kann mit K^B_{pub} eine Nachricht verschlüsseln und behaupten Alice zu sein.

Alice $\longrightarrow$ Bob : $\mathrm{E}_{\mathrm{K}^B_{priv}}[M]$

- Gewährleistet Authentifikation und Signatur
  - Nur Alice besitzt den Key K^A_{priv} zum Entschlüsseln
  - Die Nachricht wurde während der Übertragung nicht verändert
  - Jeder Kommunikationsteilnehmer, der Zugriff auf den Public Key von Alice hat, kann die Signatur verifizieren.

Alice $\longrightarrow$ Bob : $\mathrm{E}_{\mathrm{K}^B_{pub}}[\mathrm{E}_{\mathrm{K}^B_{priv}}(M)]$

- Liefert Vertraulichkeit aufgrund der Verschlüsselung mit K^B_{pub}
- Gewährleistet Authetifikation und Signatur durch $\mathrm{K}^A_{priv} i$

Tabelle 10.1: Implikationen der Nachrichtenverschlüsselung hinsichtlich Vertraulichkeit und Authentifizierung.

10.2.2 Message Authentication Code

Ein alternatives Authentifikationsverfahren setzt einen geheimen Schlüssel ein, um einen kleinen Datenblock fester Länge zu erzeugen. Diesen Datenblock nennt man **kryptographische Prüfsumme** oder **Message Authentication Code** oder kurz **MAC**. Diese Prüfsumme wird an die eigentliche Nachricht angehängt. Diese Technik setzt voraus, dass Alice und Bob einen geheimen Schlüssel K teilen. Wenn Alice eine Nachricht an Bob senden möchte, berechnet sie den MAC als Funktion der Nachricht und des Schlüssels:

$$\mathrm{MAC} = \mathrm{C_K}(M)$$

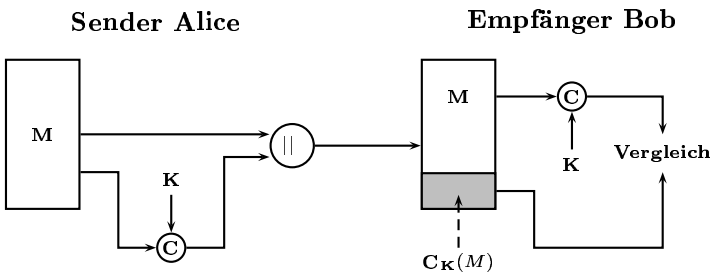


Abbildung 10.5: Grundprinzip des Message Authentication Codes (MAC) zur Authentifizierung von Nachrichten.

Die Nachricht und der Message Authentication Code werden an den Empfänger übertragen. Der Empfänger führt die gleiche Berechnung mit dem geheimen Schlüssel K wie der Sender an der erhaltenen Nachricht aus und erhält so einen zweiten Message Authentication Code. Der empfangene MAC wird mit dem berechneten verglichen (siehe Abbildung [10.5]). Wenn angenommen wird, daß nur Alice und Bob den Schlüssel K kennen und beide MACs stimmen überein, dann impliziert dies:

1. Der Empfänger der Nachricht kann sicher sein, dass diese Nachricht während der Übertragung nicht geändert wurde. Hierdurch ist also die *Integrität* gewährleistet. Falls ein Angreifer die Nachricht geändert hat, jedoch nicht den übertragenen MAC, dann unterscheiden sich der auf Empfängerseite neu berechnete und der erhaltene MAC. Da angenommen wird, dass der Angreifer den geheimen Schlüssel nicht kennt, ist er nicht in der Lage, den MAC so zu ändern, dass dieser mit der geänderten Nachricht korrespondiert.

2. Der Empfänger kann sicher sein, dass die Nachricht von dem angeblichen Sender stammt. Da kein anderer den geheimen Schlüssel kennt, kann auch niemand sonst den zur Nachricht passenden Message Authentication Code generiert haben.

3. Falls die Nachricht eine Sequence Nummer enthält – dies ist beispielsweise bei den Netzwerkprotokollen HDLC (High Level Data Link Control), X.25 und Transmission Control Protocol (TCP) der Fall – dann kann der Empfänger sicher sein, dass die Reihenfolge der empfangenen Pakete nicht verändert wurde. Denn ein Angreifer kann nicht die Sequence Nummer ändern, ohne Änderung des Message Authentication Codes.

Eine MAC Funktion – i.e. eine Funktion, die einen Message Authentication Code generiert – ist ähnlich der einer Verschlüsselungsfunktion. Ein Unterschied besteht darin, dass der MAC Algorithmus nicht umkehrbar sein muß wie es für die Dechiffrierung der Fall ist. Aufgrund der mathematischen Eigenschaften der Authentifikationsfunktion hat sich herausgestellt, dass diese weniger angreifbar sind als Verschlüsselungsfunktionen.

Das oben beschriebene Verfahren gewährleistet die Authentifikation einer Nachricht jedoch nicht die Vertraulichkeit, da die Nachricht in Klartextform übertragen wird. Die Vertraulichkeit wird durch Verschlüsselung gewährleistet. Diese kann vor oder nach Anwendung der MAC Erzeugung durchgeführt werden. In jedem Fall sind zwei verschiedene Schlüssel erforderlich, die Sender und Empfänger kennen müssen.

Im ersten Fall – i.e. Erzeugung des MAC und anschließender Verschlüsselung – wird zunächst der MAC generiert indem die Authentifikationsfunktion auf die Nachricht im Klartext angewendet wird. Der MAC wird anschließend mit der Nachricht konkateniert. Der komplette Block wird anschließend in einem zweiten Schritt verschlüsselt.

Im zweiten Fall wird zuerst die Nachricht chiffriert. Anschließend wird der MAC berechnet, indem der Chiffretext durch die Authentifikationsfunktion läuft. Der auf diese Weise generierte MAC wird an den Chiffretext angehängt und an den Empfänger übertragen. Üblicherweise wird das erste Szenario präferiert, i.e. die direkte Verknüpfung des MACs mit dem Klartext.

Da bereits konventionelle, symmetrische Kryptosysteme die Authentifikation gewährleisten und solche Systeme weit verbreitet sind, stellt sich die Frage, warum diese Verfahren nicht verwendet werden und die Authentifikation von Nachrichten in einem separaten Schritt in Form des Message Authentication Codes durchgeführt wird. Es gibt eine Reihe von Szenarien, in denen es zweckmäßig ist, Verschlüsselung und Authentifizierung zu trennen:

① Es gibt Applikationen, in denen die gleiche Nachricht an mehrere Empfänger versendet wird (Broadcast). Beispiele sind Server Mitteilungen, dass der Server zu Wartungszwecken heruntergefahren wird oder ein Alarmsignal in einem militärischen Kontrollzentrum. Es ist vom Ressorcenverbrauch günstiger und außerdem zuverlässiger, die Überwachung der Authentizität der Messages von einer Station durchzuführen als diese Aufgabe auf allen Stationen zu verteilen. Daher muß die Servernachricht im Klartext mit einem anhängenden MAC gesendet werden. Die verantwortliche Station verfügt über den geheimen Schlüssel und führt die Authentifikation der Servernachricht aus. Falls dieser dedizierte Rechner eine Unregelmäßigkeit bei der Authentifizierung feststellt, sendet er eine Broadcastmeldung an alle anderen Rechner.

② Ein weiteres mögliches Szenario besteht in einem Nachrichtenaustausch, bei dem eine der Parteien eine sehr hohe Ressourcenauslastung hat und nicht die Ressourcen zur Verfügung hat, alle einlaufenden Nachrichten(-pakete) zu entschlüsseln. In diesem Fall wird die Authentifizierung auf selektiver Basis ausgeführt, indem zufällig einige der einlaufenden Pakete geprüft werden.

③ Die Authentifikation eines Computerprogramms im Klartext ist ein attraktives Feature. Das auf einem Server zentral hinterlegte Programm kann ausgeführt werden, ohne dieses bei jedem Aufruf dechiffrieren zu müssen, was einen hohen Ressourcenverbrauch zur Folge hat. Falls jedoch ein Message Authentication Code an das Programm angehängt ist, kann bei jedem Aufruf die Integrität des Programms geprüft werden.

④ Für eine Reihe von Anwendungen spielt es keine Rolle, dass Nachrichten verschlüsselt werden, die Authentifikation ist jedoch wesentlich. Ein Beispiel ist das *Simple Network Management Protocol* Version 3 (SNMPv3) (siehe beispielsweise [93, p. 377] oder [47, pp. 553]). In diesem Protokoll wird die Funktionalität der Vertraulichkeit und der Authentifikation getrennt. Für die Sicherheit dieses Protokolls ist es wichtig, dass ein Rechner, der von einem zentralen Server aus administriert wird, die einlaufenden SNMP Nachrichten authentifizieren kann. Dies gilt insbesondere für den Fall, dass die SNMP Pakete eine Instruktion enthält, um Parameter des Systems zu ändern. Andererseits ist es nicht erfoderlich, diese SNMP Pakete zu chiffrieren.

⑤ Die Trennung von Authentifikation und Verschlüsselung erforderteine Flexibilität in der Architektur der Anwendung. Beispielsweise ist es unter Umständen wünschenswert, die Authentifikation in einem Protokoll der Application Layer des ISO/OSI Modells durchzuführen, die Verschlüsselung jedoch in einem Protokoll der tieferliegenden Transport- oder Netzwerkschicht.

⑥ Ein Benutzer möchte die Verifikation der Authentizität seines Dokumentes über den Zeitraum der Übertragung und des Empfangs hinaus verlängern. Mit der reinen Verschlüsselung geht diese Eigenschaft in dem Moment verloren, wenn der Empfänger die Nachricht dechiffriert. Die Nachricht ist daher gegen betrügerische Modifikation nur während der Übertragung geschützt aber nicht auf dem Zielsystem.

Schließlich ist nochmals anzumerken, daß der MAC keine digitale Signatur gewährleistet, da Sender und Empfänger den gleichen Schlüssel besitzen. Die Tabelle [10.2] faßt nochmals die Implikationen hinsichtlich Vertraulichkeit und Authentifizierung für die verschiedenen Alternativen zusammen.

1. Alice $\longrightarrow$ Bob : $M \,||\, \mathrm{C}_K(M)$
   - Gewährleistet Authentifikation
     - Nur Alice und Bob kennen den geheimen Schlüssel K
2. Alice $\longrightarrow$ Bob : $\mathrm{E}_{K_2}[M \,||\, \mathrm{C}_{K_1}(M)]$
   - Gewährleistet Authentifikation
     - Ausschließlich Alice und Bob kennen den geheimen Schlüssel K_1
   - Gewährleistet Vertraulichkeit
     - Der geheime Schlüssel K_2 ist nur Alice und Bob bekannt
3. Alice $\longrightarrow$ Bob : $\mathrm{E}_{K_2}[M] \,||\, \mathrm{C}_{K_1}(\mathrm{E}_{K_2}[M])$
   - Gewährleistet Authentifikation
     - Verwendung von K_1
   - Gewährleistet Vertraulichkeit
     - Verwendung von K_2

Tabelle 10.2: Grundlegende Einsatzmöglichkeiten von MACs.

10.2.3 Hash Funktionen

Eine Variante des Message Authentication Codes ist die sogenannte **One-Way Hash Funktion**. Ähnlich dem Message Authentication Code akzeptiert eine Hash Funktion eine Nachricht M variabler Länge und produziert einen Hash-Wert $H(M)$ fester Länge als Ausgabe. Dieser Hash-Wert wird auch **Message Digest** genannt. Der Hash Code ist eine Funktion sämtlicher Nachrichtenbits und liefert eine zusätzliche Möglichkeit der Fehlererkennung. Eine Änderung eines einzelnen Bits der Nachricht hat eine Änderung des Hash Codes zur Folge.

Die Abbildungen [10.6] bis [10.11] stellen eine Vielzahl von Möglichkeiten dar, wie Hash Codes eingesetzt werden können, um Message Authentifizierung zu realisieren.

❶ **Version 1:**
Die Nachricht wird zusammen mit dem konkatenierten Hash Code mit einem symmetrischen Verschlüsselungssystem chiffriert. Diese Methode ist von der Struktur her identisch mit der Strategie der internen Fehler-Kontrolle[2]. In dieser Version kann die gleiche Argumentation wie zuvor durchgeführt werden: Da nur Alice und Bob den Schlüssel K kennen, muß die Nachricht von Alice kommen und wurde während der Übertragung nicht verändert. Der Hash Code liefert die erforderliche Struktur oder Redundanz, die Authentifizierung umzusetzen. Da die Nachricht und der Hash Code verschlüsselt werden, liefert diese version auch Vertraulichkeit.

[2] Darunter versteht man beispielsweise die Verwendung einer CRC Codierung beispielsweise im 802.11 WLAN Standard zur Fehlererkennung.

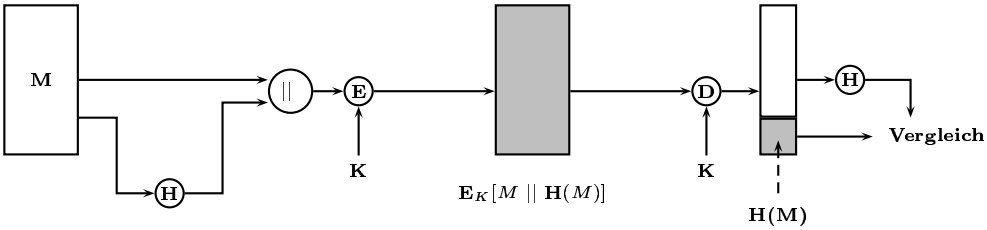


Abbildung 10.6: Methode der Hash Funktion Version 1. Der Message Digest wird mit der Nachricht konkateniert (Symbol ||). Beide Blöcke werden mit dem geheimen Schlüssel K (symmetrisches Kryptosystem) verschlüsselt und an Bob übertragen.

❷ **Version 2:**
In der Version 2 wird nur der Hash Code mit Hilfe eines symmetrischen Kryptosystems chiffriert. Dieses Verfahren ist in der Abbildung [10.7] dargestellt. Wird diese Variante eingesetzt, dann hat dies den Vorteil, dass die Prozessorlast für den Empfänger beträchtlich reduziert wird. Daher ist diese Version für Anwendungen geeignet, die keine Vertraulichkeit erfordern. Die Kombination von Hashing und Verschlüsselung liefert eine Funktion, die ein Message Authentication Code ist (siehe dazu auch die Abbildung [10.5]). Das bedeutet,

$$\mathrm{E_K}[H(M)]$$

ist eine Funktion der Nachricht M mit variabler Länge und dem geheimen Schlüssel K. Dies liefert einen Output fester Länge, der sicher ist gegen einen Angreifer, der den geheimen Schlüssel nicht kennt.

❸ **Version 3:** In der dritten Version wird wiederum nur der Hash Code chiffriert. In diesem Fall wird jedoch ein Public-Key Kryptosystem eingesetzt, daher wird mit dem Private Key des Senders verschlüsselt. Wie Version 2 gewährleistet dieses Verfahren Authentifizierung. Version 3 liefert jedoch auch digitale Signatur, da ausschließlich der Sender der Nachricht den Private Key besitzt. Diese Version ist in der Praxis die Grundlage der **digitalen Signatur**.

❹ **Version 4:**
Wenn für eine Kommunikation sowohl Vertraulichkeit als auch digitale

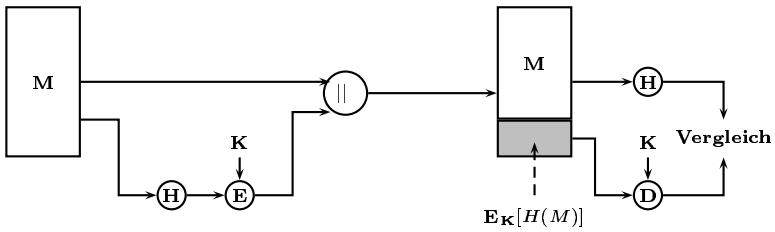


Abbildung 10.7: Version 2: Einsatzmöglichkeit von Hash Funktionen. Der Message Digest wird mit Hilfe eines geheimen Schlüssels chiffriert und anschließend an die Nachricht gehängt. Diese Version führt keine Chiffrierung der Nachricht durch.

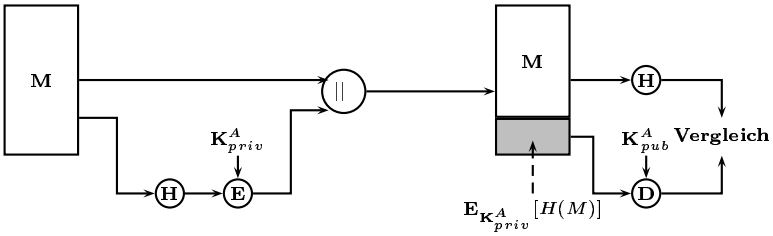


Abbildung 10.8: Version 3: Einsatz von Hash Funktionen zur Realisierung der digitalen Signatur.

Signatur erforderlich ist, kann sowohl der mit einem Public-Key Kryptosystem verschlüsselte Hash Code und die Nachricht mit einem symmetrischen Kryptosystem chiffriert werden. Diese Methode ist in der Abbildung [10.9] dargestellt.

❺ Version 5:
Die Version 5 – siehe Abbildung [10.10] – verwendet eine Hash Funktion aber keine Verschlüsselung der Message Authentication. Diese Technik setzt voraus, daß die beiden Kommunikationsteilnehmer ein gemeinsames Geheimnis S kennen. Alice berechnet den Hash Wert über die Konkatenation von M und S und hängt den so ermittelten Hash Code an die Nachricht M. Da Bob den Wert S kennt, kann er den Hash Wert erneut berechnen und vergleichen. Da der geheime Wert S nict übertragen wird, kann ein Angreifer eine abgefangene Nachricht nicht modifizieren und kann keine falsche Nachricht erzeugen.

❻ Version 6:

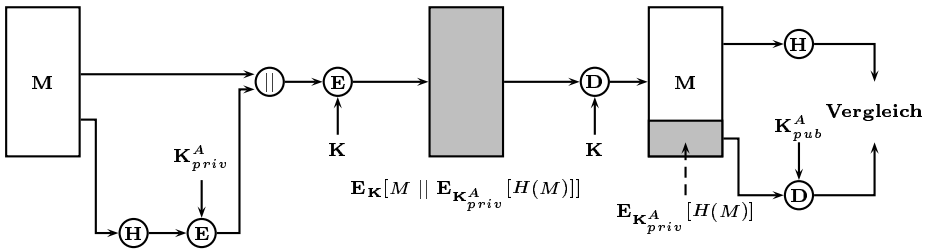


Abbildung 10.9: Version 4: Realisierung von Vertraulichkeit und digitaler Signatur

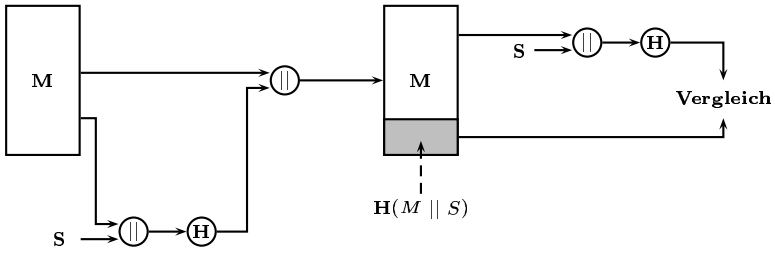


Abbildung 10.10: Version 5: Authentifikation mit Hash Funktionen

Um in der in Version 5 diskutierten Variante auch Vertraulichkeit zu gewährleisten, wird die komplette Nachricht und der Hash Code chiffriert.

Falls die Vertraulichkeit nicht erfoderlich ist, haben die in Version 2 und 3 dargestellten Methoden einen Vorteil gegenüber denjenigen, die gesamte Nachricht verschlüsseln. Da nur der Hash Wert chiffriert wird, erfodern diese beiden Methoden beträchtlich weniger Rechenaufwand. Nichtsdestotrotz besteht wachsendes Interess in Techniken, die die verschlüsselung vermeiden. (Abbildung [10.10]). Dafür gibt es mehrere Gründe:

- Verschlüsselungssoftware ist verhältnismäßig langsam. Obwohl die zu verschlüsselnde Datenmenge pro Mail im Durchschnitt relativ gering ist, kann ein permanenter Nachrichtenstrom mehrere Mails aus einem System heraus oder in ein System hinein vorliegen.
- Die Kosten für Verschlüsselungshardware sind in der Regel nicht vernachlässigbar. Preiswerte Chips, auf denen der DES Algorithmus implementiert ist sind verfügbar. Die Kosten summieren sich jedoch bei großen lokalen Netzwerken.
- Verschlüsselungshardware ist optimiert für die Verarbeitung großer Datenmengen. Bei der Verarbeitung kleiner Datenblöcke wird ein hoher Anteil

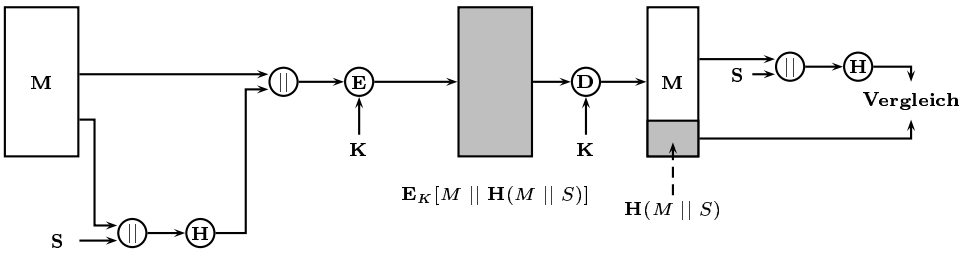


Abbildung 10.11: Version 6: Hash Funktionen, Vertraulichkeit und Authentifikation.

der Zeit für Initialisierung und Aufruf der Einheit verbraucht.

- Verschlüsselungssoftware kann mit Patenten belegt sein. Bis in das Jahr 2000 war beispielsweise der RSA Algorithmus patentiert und für dessen Verwendung in kryptographischen Protokollen wurden Lizenzgebühren erhoben. Dies führt unter Umständen zu weiteren Kosten.
- Verschlüsselungsalgorithmen unterliegen US Export Restriktionen.

10.3 Weiterführendes über Hash Funktionen

Ein **Hash Wert** wird durch eine Funktion

$$h = H(M)$$

erzeugt. Hierbei ist M eine Nachricht variabler Länge und $H(M)$ ist der Hash Wert fester Länge. Der Hash Wert wird durch den Sender an eine Nachricht angehängt. Dies geschieht zu einem Zeitpunkt, wenn angenommen werden kann, daß die Nachricht vollständig und korrekt ist. Der Empfänger authentifiziert die Nachricht indem er den Hash Wert erneut berechnet. Da die Hash Funktion nicht geheim gehalten werden soll (KERCKHOFFS Prinzip) sind eine Reihe von Maßnahmen erforderlich, den Hash Wert zu schützen.

10.3.1 Generelle Anforderungen an Hash Funktionen

Sinn und Zweck einer Hash Funktion ist die Generierung eines 'Fingerabdrucks' einer Datei, einer Nachricht oder eines Datenblocks. Um für den Einsatz in Authentifikationsverfahren verwendet werden zu können, sind an die Hash Funktion eine Reihe von Anforderungen zu stellen:

① H kann auf einen Block beliebiger Größe angewendet werden.

② H erzeugt einen Output fester Größe.

③ $H(x)$ ist für jedes x relativ leicht zu berechnen. Dies ist die Voraussetzung für effektive Implementierung in Form von Hard- oder Software.

④ Für einen gegebenen Hash Wert ist es rechnerisch undurchführbar, den Wert x zu finden, so daß $h = H(x)$. Diese Eigenschaft nennt man **One-Way Eigenschaft**. Daher betrachtet man in der Regel **One-Way Hash Funktionen**.

⑤ Für einen vorgegebenen Block x ist es rechnerisch undurchführbar, einen Block $y \neq x$ zu finden, so daß $H(x) = H(y)$. Diese Forderung nennt man **schwache Kollisionsresistenz**.

⑥ Es ist rechnerisch nicht durchführbar, ein Blockpaar (x, y) zu finden, so daß $H(x) = H(y)$ gilt. Diese Forderung nennt man **starke Kollisionsresistenz**.

Anmerkungen:

✎ Die ersten drei Forderungen resultieren aus der praktischen Anwendung von Hash Funktionen zur Nachrichten Authentifizierung.

✎ Die vierte Anforderung ist die One-Way Eigenschaft. Es ist also vom Rechenaufwand her einfach, einen Hash Code zu erzeugen, wenn die Nachricht gegeben ist. Umgekehrt ist es praktisch nicht möglich, von einem bekannten Hash Wert auf die Nachricht zu schließen. Diese Eigenschaft einer Hash Funktion ist essentiell für das ganze Verfahren, wenn das Authentifikationsverfahren die Verwendung eines geheimen Wertes vorsieht (siehe Abbildung [10.10]). Der geheime Wert selbst wird nicht übertragen.

Falls aber die Hash Funktion nicht One-Way ist, kann ein Angreifer den geheimen Wert – den wir im folgenden S_{AB} nennen – auf einfache Weise rekonstruieren. Dies kann folgendermaßen eingesehen werden:

Falls der Angreifer eine Übertragung abhört und in den Besitz der Nachricht M und des Hash Codes

$$\mathrm{C} = \mathrm{H}(S_{AB} \,||\, M)$$

gelangt, dann invertiert der Angreifer[3] die Hash Funktion und erhält:

$$S_{AB} \,||\, M = \mathrm{H}^{-1}(\mathrm{C}).$$

Da der Angreifer nun im Besitz von M und $S_{AB} \,||\, M$ ist, ist es trivial daraus deen geheimen Wert S_{AB} abzuleiten.

- Die fünfte Eigenschaft garantiert, daß keine zweite Nachricht gefunden werden kann, die auf den gleichen Hash Wert führt. Dies verhindert die Fälschung eines Dokumentes, wenn ein verschlüsselter Hash Code eingesetzt wird (wie in den Versionen [10.7] und [10.8]). In diesen Fällen hat der Angreifer Zugriff auf den Klartext und kann daher den Hash Code ebenfalls generieren. Da der Angreifer jedoch nicht im Besitz des geheimen Schlüssels ist, kann er die Nachricht nicht unbemerkt modifizieren. Falls eine Hash Funktion diese fünfte Eigenschaft nicht erfüllt, kann ein Angreifer die folgenden Schritte ausführen:
  - Zuerst wird eine Nachricht mit dem angehängten, verschlüsselten Hash Code abgefangen.
  - Erzeuge in einem zweiten Schritt einen nicht chiffrierten Hash Code aus der abgefangenen Nachricht.
  - Drittens: Generiere eine andere Nachricht mit dem gleichen Hash Code.
- Die sechste Eigenschaft ist erforderlich, damit eine Hash Funktion resistent gegen eine Angriffsart ist, die man **Geburtstagsangriff** nennt. Diese Angriffsart betrachten wir in Abschnitt [10.3.3] im Detail.

10.3.2 Einfache Hash Funktionen

Sämtliche in Authentifizierungsprotokollen eingesetzte Hash Funktionen operieren nach den folgenden Prinzipien: Der Input, i.e. eine Nachricht, eine Datei, etc., wird als Folge von Bit Blöcken behandelt. Jeder Block hat die feste Länge von n Bit. Der Input wird iterativ blockweise verarbeitet und produziert einen n-Bit Hash Wert.

Eine der einfachsten Hash Funktionen ist die bitweise XOR Verknüpfung der Inputblöcke. Dieses Verfahren kann quantitativ folgendermaßen ausgedrückt werden:

$$C_i = b_{i1} \oplus b_{i2} \oplus \cdots \oplus b_{im}$$

[3] Wir nehmen an, daß dies einfach durchzuführen ist.

wobei

$$\begin{aligned} C_i &= i\text{tes Bit des Hash codes}, \quad 1 \le i \le n \\ m &= \text{Anzahl der } n \text{ Bit Blöcke im Input} \\ b_{ij} &= i\text{tes Bit des } j\text{-ten Blocks} \\ \oplus &= \text{XOR Operation} \end{aligned}$$

In der Tabelle [10.3] ist das Schema dieser Operation dargestellt. Dieses Verfahren erzeugt eine einfache Parität für jede Bitposition. Diese einfache Hash Funktion ist unter dem Begriff *longitudinal redundancy check* bekannt. Diese Hash Funktion ist zweckmäßig für zufällige Datenbits zur Integritätsprüfung. In diesem Fall hat jeder n Bit Hash Wert die gleiche Wahrscheinlichkeit. Daher ist die Wahrscheinlichkeit, daß ein Übertragungsfehler[4] auf einen unveränderten Hash Wert führt 2^{-n}. Für strukturierte, formatierte Daten ist diese Hash Funktion jedoch weniger nützlich. Beispielsweise hat in normalen ASCII Textdateien das höchstwertige Bit jedes Oktetts den Wert `0`[5]. Falls daher ein 128-Bit Hash Wert verwendet wird, liegt die Effektivität anstatt 2^{128} lediglich bei einen Wert von 2^{112}.

	Bit 1	**Bit 2**	$\cdots$	**Bit n**
Block 1	b_{11}	b_{21}	$\cdots$	b_{n1}
Block 2	b_{12}	b_{22}	$\cdots$	b_{n2}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
Block m	b_{1m}	b_{2m}	$\cdots$	b_{nm}
Hash Code	C_1	C_2	$\cdots$	C_n

Tabelle 10.3: Einfache Hash Funktion mit bitweiser XOR Verknüpfung.

Beispiel:

Wir betrachten den folgenden Bit String:

```
0100 1101 1011 0101 1000
```

In diesem Beispiel besteht die mit Hilfe eines Hash Codes zu authentifizierende Nachricht aus fünf Blöcke mit jeweils vier Bits.

	Bit 1	**Bit 2**	**Bit3**	**Bit 4**
Block 1	0	1	0	0
Block 2	1	1	0	1
Block 3	1	0	1	1
Block 4	0	1	0	1
Block 5	1	0	0	0
Hash Code	$C_1 = 1$	$C_2 = 1$	$C_3 = 1$	$C_4 = 1$

[4] In diesem Kontext versteht man unter einem Fehler die Änderung eines Bits.

[5] Der Grund dafür ist, daß im ASCII Code Kleinbuchstaben die ASCII Werte `097` bis `122` haben, die Großbuchstaben liegen im Wertebereich `065` bis `090`. Im erweiterten, 8-Bit ASCII Code sind dies alles Zahlenwerte mit einer führenden `0`.

Daher lautet der Hash Code dieses Beispiels

$$\texttt{C = 1111}$$

Eine einfache Möglichkeit, die Effektivität der Hash Funktion zu verbessern besteht darin, einen 1-Bit zirkulären Left Shift – oder auch *Rotation* genannt – auf den Hash Wert durchzuführen, nachdem jeder Block verarbeitet wurde. Dieses Verfahren kann folgendermaßen dargestellt werden:

1. Initialisiere den n Bit Hash Wert mit 0.
2. Jeder nachfolgende n Bit Datenblock wird folgendermaßen verarbeitet:
   (a) Rotiere den aktuelle Hash Wert um ein Bit nach links
   (b) XOR den Block mit dem aktuellen Hash Wert.

Diese Modifikation der Hash Funktion hat den Effekt, den Input über den Hash Wert zufälliger zu verteilen. Dadurch werden Regularitäten des Inputs über den resultierenden Hash Wert zu verteilen.

Beispiel:

Wir betrachten nochmals den Bitstring:

$$\texttt{0100 1101 1011 0101 1000}$$

mit den fünf Datenblöcken

$$\begin{aligned} B_1 &= \texttt{0100} \\ B_2 &= \texttt{1101} \\ B_3 &= \texttt{1011} \\ B_4 &= \texttt{0101} \\ B_5 &= \texttt{1000} \end{aligned}$$

Der erste Step des Algorithmus besteht in der Initialisierung des Hash Wertes mit 0. i.e.

$$C_0 = \texttt{0000}$$

Die Rotation des aktuelle Hash Wertes um eine Stelle nach links führt zu keiner Änderung:

$$C_0' = \texttt{0000}$$

Nun führen wir die XOR Verknüpfung des ersten Datenblocks mit dem aktuellen Hash Wert durch:

$$\begin{aligned} C_0' \oplus B_1 &= (\texttt{0000}) \oplus (\texttt{0100}) \\ &= (\texttt{0100}) \\ &= C_1 \end{aligned}$$

Diese Bitfolge wird nun um ein Bit nach links rotiert:

$$C_1' = \texttt{1000}$$

In der zweiten Runde wird dieser Bit Block mit dem zweiten Datenblock B_2 geXORt:

$$\begin{aligned} C'_1 \oplus B_2 &= (1000) \oplus (1101) \\ &= (0101) \\ &= C_2 \end{aligned}$$

Dieser Hash Wert wird nun um ein Bit nach links verschoben:

$$C'_2 = 1010$$

Nächste Runde:

$$\begin{aligned} C'_2 \oplus B_3 &= (1010) \oplus (1011) \\ &= (0001) \\ &= C_3 \\ &\Longrightarrow C'_3 = 0010 \end{aligned}$$

und:

$$\begin{aligned} C'_3 \oplus B_4 &= (0010) \oplus (0101) \\ &= (0111) \\ &= C_4 \\ &\Longrightarrow C'_4 = 1110 \end{aligned}$$

und schließlich:

$$\begin{aligned} C'_4 \oplus B_5 &= (1110) \oplus (1000) \\ &= (0110) \\ &= C_5 \end{aligned}$$

Obwohl die zweite Variante ein geeignetes Verfahren ist, um Daten*integrität* zu gewährleisten, ist es tatsächlich ungeeignet, um Daten*sicherheit* zu realisieren, wenn ein verschlüsselter Hash Code eingesetzt wird wie in den Versionen der Abbildungen [10.7] oder [10.8].

Gegeben sei irgendeine Nachricht, dann ist es mit dieser Technik sehr einfach eine zweite Nachricht zu erzeugen, die auf den gleichen Hash Wert führt: Man erstellt einfach die gewünschte Nachricht und fügt einen n Bit Block hinzu, der den Effekt hat, daß die neue Nachricht zusammen mit dem geeigneten gewählten Block auf den gegebenen Hash Code führt.

Obwohl eine einfache XOR oder rotierte XOR (RXOR) Operation nicht zur Authentifikation ausreicht – genauer nicht die erforderliche Sicherheit bietet – scheint dieses Verfahren dann brauchbar zu sein, wenn in dem Authentifikationsverfahren die Nachricht und der Hash Code verschlüsselt werden. Dies ist die in der Abbildung [10.6] dargestellte Version. Dies ist jedoch nur mit äußester Vorsicht zu gebrauchen. Ein Verfahren, das ursprünglich vom NIST vorgeschlagen wurde, wendet die einfache XOR Operation auf 64-Bit Blöcke des Klartexts

an. Anschließend wird die komplette Nachricht verschlüsselt und der Cipher Block Chaining Modus angewendet (CBC).

Dieses Verfahren kann folgendermaßen definiert werden:

- Gegeben ist eine Nachricht M, die aus einer Folge von 64 Bit Blöcken $\mathtt{X}_1, \mathtt{X}_2, \ldots \mathtt{X}_N$ besteht.
- Der Hash Code C ist definiert als Block-für-Block XOR Verknüpfung aller Datenblöcke:

$$C = \mathtt{X}_{N+1} = \mathtt{X}_1 \oplus \mathtt{X}_2 \oplus \cdots \oplus \mathtt{X}_N$$

- Dieser Hash Code C wird als letzter Block $\mathtt{X}_{N+1}$ an die Nachricht angehängt.
- Die komplette Nachricht mit konkateniertem Hash Code wird im CBC Modus mit einer symmetrischen Chiffre verschlüsselt und liefert die Chiffreblöcke

$$\mathtt{Y}_1, \mathtt{Y}_2, \ldots, \mathtt{Y}_{N+1}$$

Der Chiffretext dieser Nachricht kann derart manipuliert werden, daß diese Veränderung nicht vom Hash Code bemerkt wird. Beispielsweise haben wir per Definition des CBC[6]:

$$\begin{aligned} \mathtt{X}_1 &= IV \oplus D_K(\mathtt{Y}_1) \\ \mathtt{X}_i &= \mathtt{Y}_{i-1} \oplus D_K(\mathtt{Y}_i) \\ \mathtt{X}_{N+1} &= \mathtt{Y}_N \oplus D_K(\mathtt{Y}_{N+1}) \end{aligned}$$

Nun ist aber $\mathtt{X}_{N+1}$ der Hash Code

$$\begin{aligned} \mathtt{X}_{N+1} &= \mathtt{X}_1 \oplus \mathtt{X}_2 \oplus \cdots \mathtt{X}_N \\ &= (IV \oplus D_K(\mathtt{Y}_1)) \oplus (\mathtt{Y}_1 \oplus D_K(\mathtt{Y}_2)) \oplus \cdots \oplus (\mathtt{Y}_{N-1} \oplus D_K(\mathtt{Y}_N)) \end{aligned}$$

Da die einzelnen Terme in der vorhergehenden Formel in beliebiger Reihenfolge XOR verknüpft werden können, folgt, daß sich der Hash Code nicht ändert, wenn die Chiffreblöcke vertauscht werden.

10.3.3 Geburtstagsangriff

In diesem Abschnitt untersuchen wir die notwendige Bedingung für die Sicherheit von Hashfunktionen, die nur auf die Anzahl der möglichen Fingerabdrücke einer Nachricht beruht. Diese notwendige Bedingung resultiert aus einer einfachen Methode, mit der Kollisionen entdeckt werden können, dem **Geburtstagsangriff**.

Angenommen, ein 64 Bit Hash Code wird eingesetzt. Naiverweise würde man glauben, dies ist eine ausreichende Länge des Message Digests, so dass keine zwei unterschiedlichen (sinnvolle) Nachrichten die gleiche 64 Bit Folge generieren. Falls ein verschlüsselter Hash Code C zusammen mit einer Klartextnachricht übertragen wird – dies entspricht den in Abbildungen [10.7] und [10.8]

[6] IV bedeutet Initial Vector.

dargestellten Szenarien – dann muß ein Angreifer einen Klartext M' finden, so daß $H(M') = H(M)$, um die echte Nachricht M durch die falsche Nachricht M' zu ersetzen und damit den Empfänger zu täuschen. Im Mittel muß der Angreifer $2^{64}/2$ Nachrichten austesten, um eine Nachricht zu finden, die den gleichen Hash Code hat wie die abgefangene Message.

Es gibt eine andere Angriffsart, die auf dem sogenannten **Geburtstags Paradoxon** basiert. Dabei wird folgende Strategie empfohlen:

① Die Nachrichtenquelle. i.e. Alice, signiert eine Nachricht, indem der geeignete m-Bit Message Authentication Code generiert wird, mit dem Private Key des Senders verschlüsselt und das Resultat dieser Operationen an die Nachricht angehängt wird. Dieser Vorgang ist in Abbildung [10.8] dargestellt.

② Der Angreifer generiert $2^{m/2}$ Varianten der Nachricht. Alle Varianten beziehen sich im wesentlichen auf den gleichen Inhalt.

The attacker generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations on the fraudulent message to be substituted for the original one.

③ The opponent now compares the two sets of messages until he finds a pair of messages that produces the same hash code. Th probabilty of success – due to the birthday paradox – is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.

④ The opponent now offers the valid variation to Alice for signature. This signature can then be attached to the fraudulent variation of the message for transmission to Bob. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is unknown.

Thus, due to the above attack, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32}.

Die Erzeugung vieler Varianten einer Nachricht mit dem gleichen Inhalt ist leicht durchführbar. Der Angreifer baut einfach eine Reihe von 'Blank-Blank-Backspace' Zeichen zwischen Worte des Dokumentes ein. Varianten davon können erzeugt werden durch ersetzen von 'Blank-Backspace-Blank' an ausgewählten Stellen. Alternativ kann der Angreifer einfach die Nachricht umformulieren ohne dabei die Bedeutung der Nachricht zu ändern.

Das Geburtstags Paradoxon

Das Geburtstags Paradoxon ist eine einfache Übung in elementarer Wahrscheinlichkeitstheorie. Dsa dem Paradoxon zugrundeliegende Problem kann folgendermaßem formuliert werden:

> *Welcher ist der minimale Wert von k, so dass die Wahrscheinlichkeit größer 0.5 ist, dass wenigstens zwei Personen in einer Gruppe von k Personen am gleichen Tag Geburtstag haben?*

Wir ignorieren den berühmten 29. Februar und nehmen an, dass alle 365 Tage des Jahres gleichwahrscheinlich sind.

Wir haben daher zu berechnen:

> $P(n,k)$ = Wahrscheinlichkeit(*wenigstens ein Duplikat in k Elementen, wobei jedes Element mit gleicher Wahrscheinlichkeit einen Wert aus der Menge 1 bis n annehmen kann*)

Wir suchen den kleinsten Wert k, so dass $P(365,k) \geq 0.5$. Wir berechnen zunächst die Wahrscheinlichkeit dafür, dass *keine* Duplikate auftreten. Diese Größe bezeichnen wir mit $Q(365,k)$. Falls $k > 365$, dann ist es unmöglich, dass keine Duplikate auftreten. Mit anderen Worten, falls wir eine Gruppe mit mehr als 365 Personen betrachten, dann müssen sicherlich mindestens zwei Personen am gleiche Tag Geburtstag haben. Daher nehmen wir an, dass $k \leq 365$. Wir betrachten nun die Anzahl der verschiedenen Möglichkeiten – diese nennen wir N – die k annehmen kann, ohne das Auftreten von Duplikaten.

- für die erste Person kann irgendeiner der 365 möglichen Tage gewählt werden.
- die zweite Person kann dann an 364 möglichen Tagen Geburtstag haben.
- und so weiter.

Damit

$$N = 365 \cdot 364 \cdot 363 \cdot \cdots \cdot (365-k+1) = \frac{365!}{(365-k)!}$$

Wir können nun die Einschränkung fallen lassen, dass es keine Duplikate gibt. Dann kann jede Person an jedem Tag des Jahres Geburtstag haben. Damit haben wir in diesem Fall 365^k Möglichkeiten.

Aus dieser Überlegung finden wir leicht die Wahrscheinlichkeit dafür, dass bei einer Menge von 365 Personen alle Personen an verschiedenen Tagen Geburtstag haben, i.e. keine Duplikate auftreten. Über die LAPLACE Wahrscheinlichkeit erhält man:

$$Q(365,k) = \frac{365!}{(365-k)!365^k}$$

Damit erhält man unmittelbar die gesuchte Wahrscheinlichkeit über das Komplementärereignis:

$$P(365,k) = 1 - Q(365,k) = 1 - \frac{365!}{(365-k)!365^k}$$

Diese Funktion ist in der Abbildung [10.12] dargestellt. Die Wahrscheinlichkeiten sind überraschend hoch. Es gilt beispielsweise:

$$P(365,23) = 0.5073$$
$$P(365,100) = 0.9999997$$

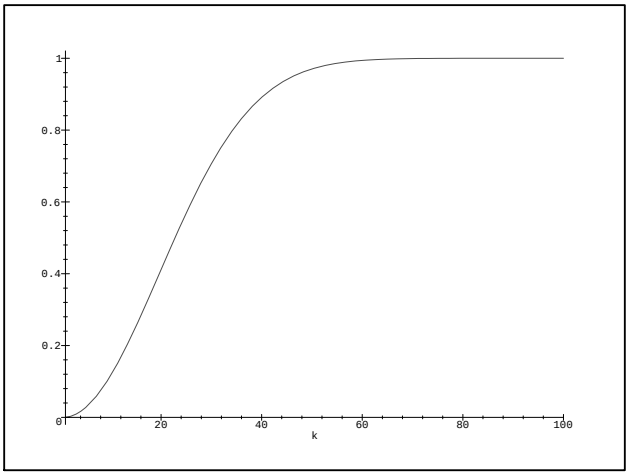


Abbildung 10.12: Der Graph der Funktion $P(365, k)$.

Der Grund für dieses überraschende Resultat liegt in folgender Überlegung: Betrachtet man eine bestimmte Person in einer Gruppe und berechnet die Wahrscheinlichkeit, dass eine andere Person dieser Gruppe am gleichen Tag Geburtstag hat, dann tritt dieses Ereignis mit geringer Wahrscheinlichkeit ein. Dies ist das Resultat, das man eigentlich erwartet. Das Ereignis, das wir hier betrachten, besteht darin, daß ein beliebiges *Paar* von Personen einer Gruppe am gleichen Tag Geburtstag hat. In einer Gruppe mit 23 Personen gibt es

$$\frac{23 \cdot (23-1)}{2} = 253$$

verschiedene Paare. Daher ist die Wahrscheinlichkeit relativ hoch, ein Paar zu finden, das am gleichen Tag Geburtstag hat.

Verallgemeinerung

Das Geburtstagsproblem kann zu folgendem Problem verallgemeinert werden:

> *Gegeben ist eine Zufallsvariable in Form einer ganzen Zahl, die gleichverteilt auf der Menge* $\{1, \ldots, n\}$ *Werte annimmt. Man wählt* k *Instanzen* ($k \leq n$) *der Zufallsvariablen aus. Wie groß ist die Wahrscheinlichkeit* $P(n, k)$ *für mindestens ein Duplikat?*

Das Geburtstagsproblem ist der spezielle Fall mit $n = 365$. Mit exakt der

gleichen Argumentation wie zuvor leitet man ab:

$$P(n,k) = 1 - \frac{n!}{(n-k)!n^k}$$

Dieser Ausdruck kann umgeschrieben werden in der Form:

$$\begin{aligned} P(n,k) &= 1 - \frac{n \cdot (n-1) \cdot \ \ldots \ \cdot (n-k+1)}{n^k} \\ &= 1 - \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \ \ldots \ \cdot \frac{n-k+1}{n} \\ &= 1 - \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdot \ \ldots \ \cdot \left(1 - \frac{k+1}{n}\right) \end{aligned}$$

Wir benutzen nun eine Ungleichung, um eine Näherung der gesuchten Wahrscheinlichkeit zu erhalten. Da[7]

$$1 - x \leq e^{-x}$$

folgt:

$$\begin{aligned} \prod_{i=1}^{n} (1 - x_i) &= (1 - x_1)(1 - x_2) \ldots (1 - x_n) \\ &\leq e^{-x_1} \cdot e^{-x_2} \cdot \ldots \cdot e^{-x_n} \\ &= \exp\{-\sum_{i=1}^{n} x_i\} \end{aligned}$$

Daher:

$$-\prod_{i=1}^{n} (1 - x_i) \geq -\exp\{-\sum_{i=1}^{n} x_i\}$$

und

$$\begin{aligned} P(n,k) &= 1 - \prod_{i=1}^{k-1} (1 - \frac{i}{n}) \\ &\geq 1 - \exp\{-\sum_{i=1}^{k-1} \frac{i}{n}\} \\ &= 1 - \exp\{-\frac{1}{n} \sum_{i=1}^{k-1} i\} \\ &= 1 - \exp\{-\frac{k(k-1)}{2n}\} \end{aligned}$$

Da die Summe der ersten n ganzen Zahlen die Summe[8]:

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}$$

[7] Dies kann mit Hilfe einer TAYLOR Entwicklung gezeigt werden.
[8] Dies zeigt man mit Hilfe der vollständigen Induktion.

ergibt, haben wir:

$$P(n,k) \geq 1 - \exp\{-\frac{k(k-1)}{2n}\} \tag{10.1}$$

Es interessiert uns nun die Frage: *Welcher Wert von k führt auf to* $P(n,k) > \frac{1}{2}$*?*
Es folgt

$$\frac{1}{2} \stackrel{!}{=} 1 - \exp\{-\frac{k(k-1)}{2n}\}$$

oder

$$2 = \exp\{\frac{k(k-1)}{2n}\}$$

Für große Werte von k können wir $k(k-1)$ durch k^2 approximieren und erhalten:

$$\ln 2 = \frac{k^2}{2n}$$

oder

$$k \approx \sqrt{2\ln(2) \cdot n} \approx \sqrt{n}$$

Als Test sollte dieses Resultat die Lösung des Geburtstagsproblems liefern. Wir setzen daher $n = 365$ und erhalten

$$k_{\text{Geburtstag}} \approx \sqrt{2 \cdot \ln(2) \cdot 365} \approx 22.5$$

was den oben abgeleiteten Wert von 23 gut annähert.

Wir sind nun in der Lage, die Grundlage des Geburtstagsangriffs zu quantifizieren:

Angenommen wir verwenden eine Funktion H mit 2^m *möglichen Ausgabewerten (i.e. ein* m*-Bit Output). Falls H angewendet wird auf k zufällige Eingabewerte, wie groß ist der Wert von k für eine gegebene Wahrscheinlichkeit, dass wenigstens ein Duplikat auftritt [i.e.* $H(x) = H(y)$ *für ein Paar* x, y*]?*

Um die Antwort zu erhalten, substitutiert man einfach:

$$k = \sqrt{2^m} = 2^{m/2}.$$

Dies bedeutet also: Bei einem m Bit Message Digest muß ein Angreifer im Schnitt $2^{m/2}$ Nachrichten produzieren, um eine Kollision zu erhalten. Mit anderen Worten (siehe [2, pp. 83-84]), Hash Funktionen, die in digitalen Signatur Schemata eingesetzt werden, müssen einen geeignet großen Hash Wert (Message Digest) m generieren, um kollisionsfrei zu sein. Das bedeutet, m muß so groß gesetzt werden, dass $2^{m/2}$ Berechnungen für einen Angreifer nicht praktikabel sind.

10.4 Der MD5 Hash Algorithmus

In der praktischen Anwendung sind eine Reihe von Hash Funktionen in Gebrauch. Die wichtigsten Algorithmen sind:

- **MD4** – Message Digest Algorithm 4
  Dieser Algorithmus wurde von **Ron Rivest** entwickelt (siehe [10.13][9]).
- **MD5** – Message Digest Algorithm 5
  MD5 wurde ebenfalls von **Ron Rivest** entwickelt, MD5 ist im wesentlichen eine Weiterentwicklung und Verbesserung von MD4.

Abbildung 10.13: Ron Rivest from MIT.

- **SHA** – Secure Hash Algorithm
  Der SHA wurde von dem NIST in Zusammenarbeit mit der NSA entwickelt und integraler Bestandteil des **Digital Signature Standards**, DSS. Am 15. Februar 2005 teilt Bruce Schneier lapidar mit: *SHA-1 has been broken. Not a reduced-round version. Not a simplified version. The real thing*[10].
- **RIPEMD-160**
  RIPEMD wurde für das RIPE Projekt der Europäischen Gemeinschaft entwickelt[11]. Der Algorithmus ist eine verbesserte Version des MD4 Verfahrens, insbesondere ist RIPEMD resistent gegen bekannte kryptoanalytische Angriffe. Der RIPEMD produziert einen 160 Bit Hash Wert.

[9] Siehe auch Ron Rivests Homapage

`http://theory.lcs.mit.edu/~rivest`

[10] Siehe:

`http://www.schneier.com/blog/archives/2005/02/sha1_broken.html`

[11] RIPE (Reseaux IP Europeens) ist eine offene Arbeitsgemeinschaft von Organisationen und Privatpersonen, die große Bereiche des europäschen Teils des Internets betreiben und

☞ **HMAC**
HMAC wurde von **H. Krawczyk, M. Bellare** und **R. Canetti**[12] beschrieben.

	MD5	SHA-1	RIPEMD-160
Digest Länge	128 bits	160 bits	160 bits
Verarbeitungseinheit	512 bits	512 bits	512 bits
Anzahl der Schritte	64 (4 x 16)	80 (4 x 20)	160 (5 paired rounds of 20)
Maximale Message Größe	∞	$2^{64}-1$ bits	$2^{64}-1$ bits
Primitive logische Funktionen	4	4	5 bits
Additive Konstante	64	4	9

Tabelle 10.4: Grundlegende Parameter der MD5, SHA-1 und RIPEMD-160 Algorithmen.

In diesem Abschnitt diskutieren wir im Detail den MD5 Algorithmus (siehe [205, pp. 272–278]).

Der MD5 Hash Algorithmus wurde von RON RIVEST am MIT entwickelt. Bis heute ist der MD5 einer der verbreiteten Hash Verfahren. Die Spezifikation von MD5 ist im RFC 1321[13] beschrieben.

Der Algorithmus hat als Input eine Nachricht beliebiger Länge und produziert einen 128 Bit Output. Dieser Output ist der Message Digest. Vor der eigentlichen Verarbeitung wird der Input in 512 Bit Blöcke aufgesplittet. Die Abbildung [10.14] zeigt die allgemeine Struktur des MD5 Algorithmus.

Die Verarbeitung der Nachricht zu einem Message Digest besteht aus folgenden fünf Schritten:

`STEP1` **Anhängen von Füllbits**
In einem ersten chritt wird die Nachricht gepaddet. Dies geschieht so, dass die Länge der Nachricht plus angehängte Bits kongruent 448 modulo 512 ist. Anders ausgedrückt, die Länge der aufgefüllten NAchricht ist 64 Bit weniger als ein Vielfaches von 512. Die verbleibenden 64 Bits werden in `STEP 2` benötigt. Das Padding wird in jedem Fall durcjgeführt, selbst wenn die Nachricht bereits die erforderliche Länge hat. Hat die Nachricht

managen. Das Ziel der RIPE Community ist die Bereitstellung administrativer und technischer Koordination zum Betrieb des europäischen Teil des Internets. RIPE unterhält kein eigenes Netzwerk.

Für weitere Informationen siehe die URL:

`http://www.ripe.net`

[12] See `RFC 2104`, H. Krawczyk, M Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, February 1997. Siehe auch [16, 17]

[13] RFC 1321; Ron Rivest, *The MD5 Message-Digest Algorithm*, April 1992.

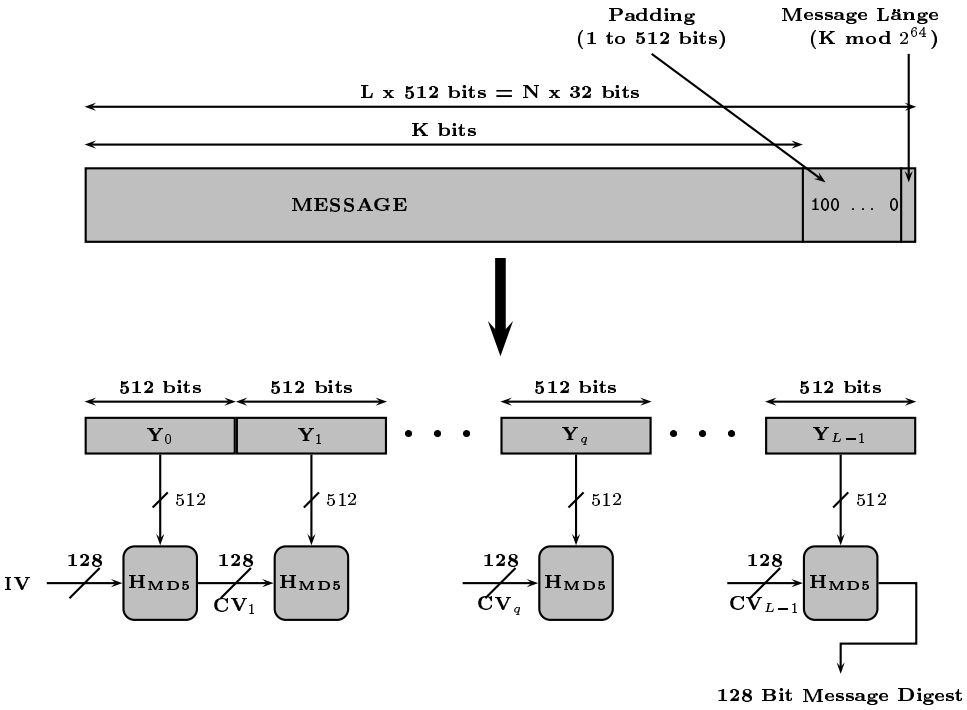


Abbildung 10.14: Erzeugung des Message Digests mit MD5. CV bezeichnet Chaining Variables.

beispielsweise die Länge von 448 Bits, dann werden 512 Bits angehängt mit einer resultierenden Gesamtlänge von 960 Bits. Daher ist die Anzahl der Füllbits aus dem Intervall [1,512]. Die Fübits bestehen aus einer führenden `1` gefolgt von der notwendigen Anzahl an `0`en.

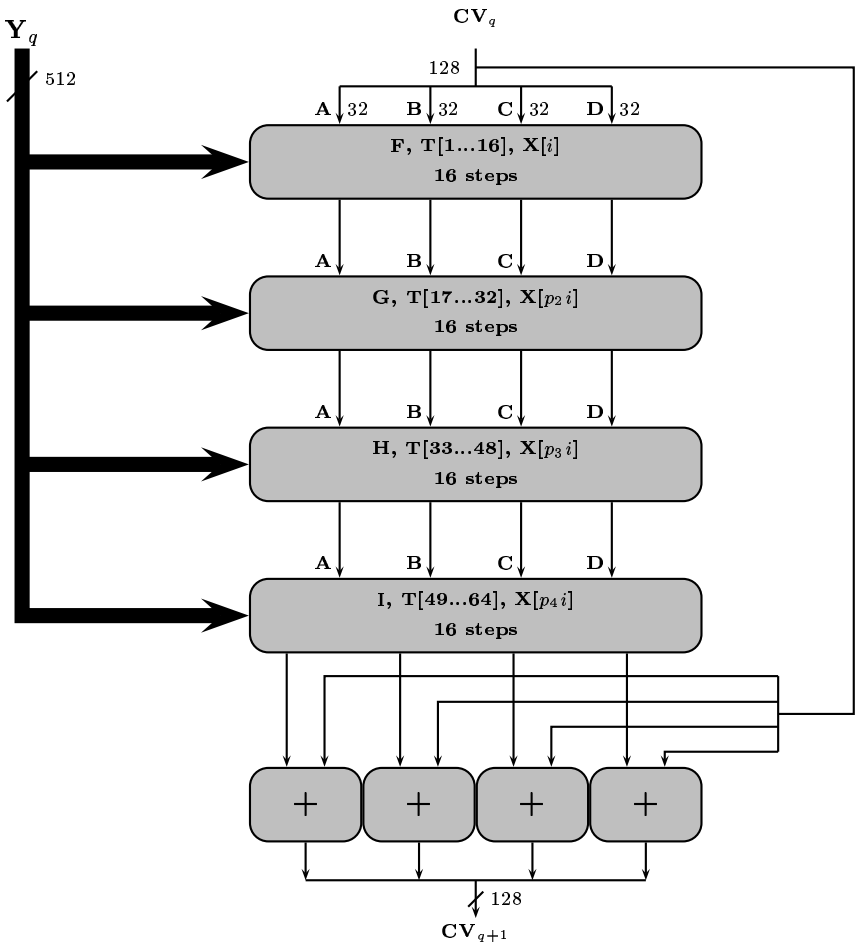


Abbildung 10.15: MD5 Processing of a Single 512 bit Block: This is the Principle of the MD5 Compression Function. Note that Addition (+) is always modulo 2^{32}.

STEP2 Länge anhängen

Im zweiten Schritt wird eine 64 Bit Darstellung der Länge in Bits der ursprünglichen Nachricht (i.e. ohne Padding) an den Output des STEPS 1 angehängt. Falls die ursprüngliche Nachricht mehr als 2^{64} Bit lang ist, werden nur die niedrigsten 64 Bits der Länge benutzt. Mit anderen Worten, der in diesem Schritt angehängte Wert ist die ursprüngliche Länge modulo 2^{64}.

Der Output der ersten beiden Schritte ist daher ein Bit String, dessen Länge ein Vielfaches von 512 Bit ist. Wie in der Abbildung [10.14] dargestellt ist, wird die ersweiterte Nachricht nun in Form von L Blöcken weiterverarbeitet.

$$\mathrm{Y}_0, \mathrm{Y}_1, \ldots, \mathrm{Y}_{L-1}$$

Jeder dieser Blöcke Y_i hat eine Länge von 512 Bit oder 16×32 Bit. Das bedeutet, der Output der ersten beiden Schritte ist eine Vielfaches von 16 32-Bit Worte. Wir bezeichnen mit $M[0, 1, \ldots, N-1]$ 32-Bit Worte der gepaddeten Nachricht, wobei N ein ganzzahliges Vielfaches von 16 ist, i.e. $N = L \times 16$.

`STEP3` **Initialisierung der Message Digest Buffers**
Ein 128 Bit Pufferspeicher wird verwendet, um Zwischenergebnisse und das Endresultat der Hash Funktion zu speichern. Der Puffer besteht aus vier 32 Bit Registern **A**,**B**,**C**,**D**. Diese Register werden mit den Werten (in Hex-Notation) vorbelegt (dies ist der Initial Vector IV in Abbildung [10.14]):

A= `67452301`
B= `EFCDAB89`
C= `98BADCFE`
D= `10325476`

`STEP4` **Verarbeitung der Message in 512-Bit (16-word) Blöcken**
Das Herz des Algorithmus ist eine **Kompressions Funktion**, die aus vier Verarbeitungsrunden besteht. In der Abbildung [10.14] entspricht dies den Blöcken, die mit H_{MD5} beschriftet sind. Die Verarbeitungslogik dieser Kompressions Funktion ist in der Abbildung [10.15] dargestellt. Alle vier Runden haben eine ähnliche Struktur, verwenden aber unterschiedliche primitive logische Verknüpfungen, die mit den Buchstaben `F, G, H` und `I` in der Abbildung [10.15] bezeichnet sind.

a	b	c	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Tabelle 10.5: Wahrheitstabelle der vier primitiven logischen Funktionen `F, G, H` und `I`.

Jede Runde hat als Input einen 512 Bit Block Y_q und den aktuellen Inhalt des 128 Bit Puffers **ABCD**. Der Inhalt des Buffers wird nach jeder Runde aktualisiert. In jeder Verarbeitungsrunde wird ein Viertel einer Tabelle `T[1,2,...,64]` mit 64 Einträgen verwendet.

Diese Tabelle wird über die Sinus Funktion erstellt. Das i-te Element der Tabelle `T` – dieses wird mit `T[i]` bezeichnet – ist gegeben durch

$$\texttt{T[i]} = \lfloor 2^{32} \times abs(\sin(i)) \rfloor$$

wobei i der Radiand ist. Mit anderen Worten `T[i]` ist der ganzzahlige Teil von $2^{32} \times abs(\sin(i))$.

Da $abs(\sin(i))$ eine Zahl zwischen 0 und 1 ist, ist jeder Eintrag der Tabelle `T` eine ganze Zahl, die durch 32 Bit dargestllt werden kann. Das Ziel bei der Verwendung solch einer Tabelle besteht darin, eine randomisierte Menge von 32 Bit Zahlen zur Verfügung zu haben, um eventuell vorhandene reguläre Strukturen der Inputdaten zu eliminieren.

Es ist nicht schwierig, die `T[i]` zu berechnen, als Beispiel führen wir dies für $i = 1, 2, 3$ durch:

$$\begin{aligned}
\texttt{T[1]} &= \lfloor 2^{32} \times \mid \sin(1) \mid \rfloor \\
&= \lfloor 2^{32} \times 0.84147098481 \rfloor \\
&= 3614090360_d \\
&= \texttt{0xD76AA478} \\
\texttt{T[2]} &= \lfloor 2^{32} \times \mid \sin(2) \mid \rfloor \\
&= \lfloor 2^{32} \times 0.90929742683 \rfloor \\
&= 3905402710_d \\
&= E8C7B756 \\
\texttt{T[3]} &= \lfloor 2^{32} \times \mid \sin(3) \mid \rfloor \\
&= \lfloor 2^{32} \times 0.14112000806 \rfloor \\
&= 606105819_d \\
&= 242070DB
\end{aligned}$$

Schließlich wird der Output der vierten Runde zum Input der ersten Runde – dies ist der Block CV_q addiert. Das Resultat ist ein Block CV_{q+1}. Die Addition wird für jedes der vier Words im Puffer **A,B,C** und **D** mit den entsprechenden Words in CV_q ausgeführt. Die Addition ist immer modulo 2^{32}.

`STEP5` **Output**

Auf diese Weise werden sämtliche L 512-Bit Blöcke verarbeitet. Das Resultat ist der 128 Bit Message Digest.

Der MD5 Algorithmus kann folgendermaßen zusammengefaßt werden:

$$\begin{aligned}
\mathrm{CV}_0 &= \mathrm{IV} \\
\mathrm{CV}_{q+1} &= \mathrm{SUM}_{32}\big(\mathrm{CV}_q, \mathrm{RF}_I[\mathrm{Y}_q, \mathrm{RF}_H[\mathrm{Y}_q, \mathrm{RF}_G[\mathrm{Y}_q, \mathrm{RF}_F[\mathrm{Y}_q, \mathrm{CV}_q]]]]\big) \\
\mathrm{MD} &= \mathrm{CV}_L
\end{aligned}$$

mit

IV	= Startwert des **ABCD** Buffers
Y_q	= der $q$te 512 Bit Block der Nachricht
L	= die Gesamtzahl der 512-Bit Blöcke in der Nachricht
CV_q	= Verkettungsvariable, die mit dem $q$ten Block der
	= Nachricht verarbeitet wird
RF_s	= Rundenfunktion, die die primitive logische Funktion
	= s verwendet
MD	= Message Digest Wert
SUM_{32}	= Addition modulo 2^{32}; wird separat auf jedes
	= der vier Worte ausgeführt

Details: MD5 Kompressions Funktion

Sehen wir uns noch die Details der Verarbeitungslogik in jeder der vier Runden an, die bei der Verarbeitung der 512 Bit Blöcke auftreten. Jede Runde besteht aus einer Folge von 16 Schritten, die auf den Inhalt des Puffers **ABCD** operieren. Jeder dieser Schritte hat die Form:

$$a \leftarrow b + ((a + x(b,c,d) + X[k] + T[i]) \lll s)$$

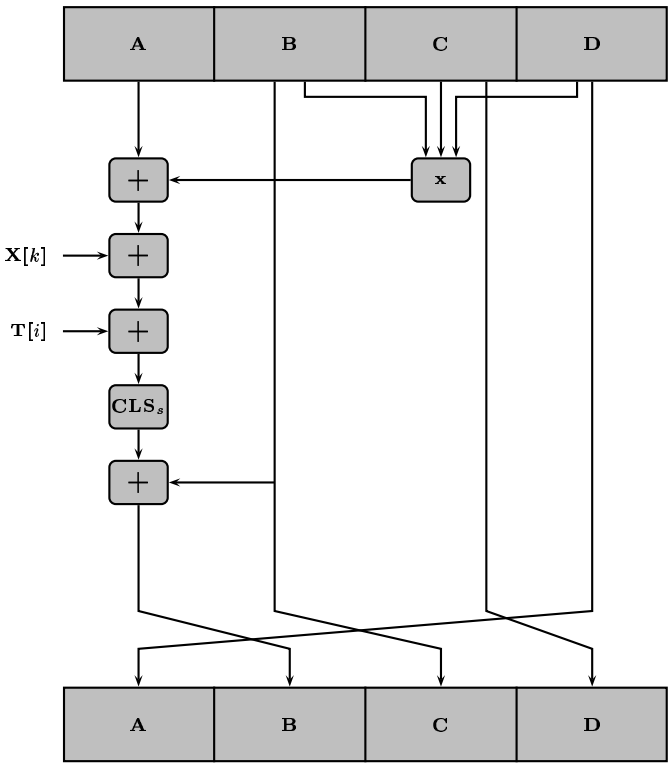


Abbildung 10.16: Elementare MD5 Operationen

mit folgender Notation:

abcd	die vier Worte des Buffers in einer festen Reihenfolge, die beim Durchlaufen der 16 Runden variieren
x	eine der vier primitiven logischen Funktionen F, G, H, I
$<<< s$	left Shift des 32-Bit Argumentes um s Bits
$X[k]$	$M[q \times 16 + k] = k$tes 32-Bit Word im $q$ten 512-Bit Block der Nachricht
$T[i]$	das $i$te 32-Bit Word in der Matrix T
$+$	Addition modulo 2^{32}

In der Abbildung [10.16] sind die Schritte dieser Operation schematisch dargestellt. Die Reihenfolge, in der die vier Words (a,b,c,d) eingestzt werden, erzeugt einen zirkulären Left Shift auf Word Level um ein Word für jeden Schritt.

Eine der vier primitiven logischen Funktionen wird für jede Runde des Algorithmus benutzt. Jede der vier Funktionen F, G, H und I bildet drei 32 Bit Worte auf ein 32 Bit Wort ab.

Round	**Function x**	**x(a,b,c)**
1	$F(a,b,c)$	$(a \wedge b) \vee (\overline{a} \wedge c)$
2	$G(a,b,c)$	$(a \wedge c) \vee (b \wedge \overline{c})$
3	$H(a,b,c)$	$a \oplus b \oplus c$
4	$I(a,b,c)$	$b \oplus (a \vee \overline{c})$

Die Symbole $\wedge, \vee, \overline{\ }, \oplus$ bezeichnen die BOOLEschen Operationen AND, OR, NOT und XOR.

a	b	c	$a \wedge b$	$\overline{a}$	$\overline{a} \wedge c$	$(a \wedge b) \vee (\overline{a} \wedge c)$
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	0
0	1	1	0	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	0	1

Tabelle 10.6: Wahrheitstabelle der primitiven logischen Funktion $F(a,b,c)$.

Die Tabelle [10.6] zeigt die Werte der primitiven logischen Funktion $F(a,b,c)$.

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and D as DD. */
  AA = A
  BB = B
  CC = C
  DD = D

  /* Round 1. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
  [ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
  [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
  [ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

  /* Round 2. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
  [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
  [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
  [ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

  /* Round 3. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
  [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
  [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
  [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

  /* Round 4. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
  [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
  [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
  [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

  /* Then perform the following additions. (That is increment each
     of the four registers by the value it had before this block
     was started.) */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD

end /* of loop on i */
```

Abbildung 10.17: MD5 Update Algorithmus (RFC 1321).

Die Abbildung [10.17] definiert die Verarbeitung in STEP4. Der Array der 32-Bit Worte $X[0 \ldots 15]$ speichert des aktuellen Wert des 512-Bit Blocks, der momentan verarbeitet wird. Innerhalb einer Runde wird jedes der 16 Worte von $X[i]$ genau einmal in einem Schritt verwendet. Die Reihenfolge, in der diese Words benutzt werden, variiert von Runde zu Runde. In der ersten Runde werden die Words in der ursprünglichen Reihenfolge verarbeitet. Die folgenden Permuta-

tionen werden für die Runden 2 bis 4 definiert:

$$\rho_2(i) = (1 + 5i) \bmod 16$$
$$\rho_3(i) = (5 + 3i) \bmod 16$$
$$\rho_4(i) = 7i \bmod 16$$

10.5 Der Secure Hash Algorithm SHA

Ein weiterer häufig genutzter Hash Algorithmus ist der **Secure Hash Algorithm**, kurz SHA. Dieser Algorithmus wurde von *National Institute of Standards and Technology* (NIST) entwickelt und im Jahre 1993 als Federal Information Processing Standard (FIPS PUB 180) publiziert. Eine revidierte Version wurde zwei Jahre später als FIPS PUB 180-1 herausgegeben; diese wird üblicherweise mit SHA-1 bezeichnet. Ähnlich wie der MD5 Algorithmus basiert SHA auf Ron Rivests MD4 Algorithmus und ähnelt in der Struktur MD5[14].

10.5.1 Die Logik des SHA-1 Algorithmus

Der SHA-1 hat als Input eine Nachricht mit maximaler Länge kleiner als 2^{64} Bits und produziert als Output einen Message Digest der Länge von 160 Bit. Wie im MD5 Algorithmus wird der Input in 512 Bit Blöcken verarbeitet.

Die grobe Struktur der Verarbeitung einer Nachricht ist beim SHA-1 Algorithmus die gleiche wie bei dem MD5 Verfahren. Dies ist in der Abbildung [10.14] dargestellt. Die Blocklänge ist 512 Bits und die Länge der Verkettungsvariablen (CV) und des Message Digest ist 160 Bit. Der SHA-1 Algorithmus kann in fünf Schritte unterteilt werden:

SHA-1 Step 1 Anhängen von Füllbits
Die Nachricht wird wie bei dem MD5 Algorithmus mit Füllbits gepaddet, so dass die daraus resultierende Länge kongruent zu 448 modulo 512 ist. Auch hier wird immer gepaddet, selbst wenn die Nachricht bereits die richtige Länge hat. Daher variiert die Anzahl der Füllbits von 1 bis 512. Das angehängte Bitmuster hat stets die Form einer führenden 1 gefolgt von der notwendigen Anzahl an 1en.

SHA-1 Step 2 Anhängen der Länge
In einem zweiten Schritt wird ein Block der Länge von 64 Bits angehängt. Der Block enthält eine unsigned 64 Bit Zahl, die die Länge der Nachricht (vor dem Padding) codiert.

SHA-1 Step 3 Initialisierung des Message Digest Buffers
SHA-1 verwendet einen 160-Bit Puffer um Zwischenresultate und das Endergebnis der Hash Funktion zwischenzuspeichern. Dieser Puffer kann durch fünf 32 Bit Register dargestellt werden, die mit `ABCDE` bezeichnet werden. Diese Register werden folgendermaßen vorbelegt (in Hex-Notation):

```
A= 67452301
B= EFCDAB89
C= 98BADCFE
D= 10325476
E= C3D2E1F0
```

[14] Der SHA-1 Algorithmus ist auch im RFC 3174 beschrieben.
RFC 3174, D. Eastlake, P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, September 2001.

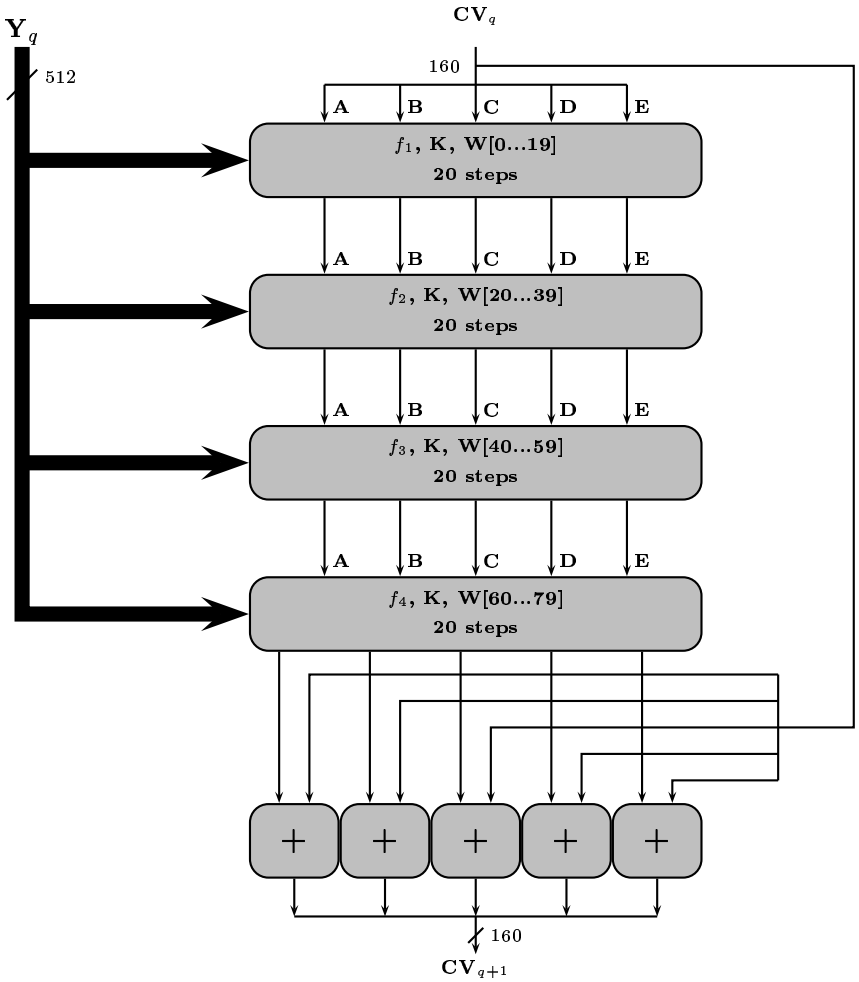


Abbildung 10.18: SHA-1: Verarbeitung eines einzelnen 512 Bit Blocks: Dies stellt das Prinzip der SHA-1 Kompressionsfunktion dar. Die Addition (+) ist hier ebenfalls immer modulo 2^{32}.

SHA-1 Step 4 Verarbeitung des Nachricht in 512 Bit Blöcken

Das Herzstück des Algorithmus ist ein Modul, das aus vier Verarbeitungsrunden besteht. Jede Runde besteht aus 20 Einzelschritten. Der grobe Ablauf dieser Komressionsfunktion ist in der Abbildung [10.18] dargestellt. Die vier Runden haben alle eine ähnliche Struktur. Jede Runde verwendet aber eine andere primitive logische Funktion, die mit f_1, f_2, f_3 und f_4 bezeichnet sind.

Der Input in jede Runde ist der aktuelle 512-Bit Block Y_q, der verarbeitet werden soll, und der 160-bit Bufferwert in `ABCDE`. Nach der Verarbeitung wird der `ABCDE` Puffer aktualisiert. Zusätzlich geht in jeder Runde eine additive Konstante K_t ein, wobei $0 \leq t \leq 79$. Effektiv werden dabei jedoch nur vier verschiedene Wert genutzt:

$$K_t = \begin{cases} 0x5A827999 & = \lfloor 2^{30} \times \sqrt{2} \rfloor \quad \text{für } 0 \leq t \leq 19 \\ 0x6ED9EBA1 & = \lfloor 2^{30} \times \sqrt{3} \rfloor \quad \text{für } 20 \leq t \leq 39 \\ 0x8F1BBCDC & = \lfloor 2^{30} \times \sqrt{5} \rfloor \quad \text{für } 40 \leq t \leq 59 \\ 0xCA62C1D6 & = \lfloor 2^{30} \times \sqrt{10} \rfloor \quad \text{für } 60 \leq t \leq 79 \end{cases}$$

Die Ausgabe der vierten Runde – nach Schritt 80 – wird zum Input der ersten Runde – i.e. CV_q – addiert und produziert dadurch den Output CV_{q+1}. Die Addition wird unabhängig voneinander für jedes der fünf 32-bit Worte `A, B, C, D,` und `E` in dem Puffer mit jedem der korrespondierenden Words in CV_q ausgeführt. Dabei wird Addition modulo 2^{32} verwendet.

SHA-1 Step 5 Output

Nachdem alle L Blöcke der Länge von 512 Bits auf diese Weise verarbeitet werden ist die Ausgabe des Schrittes L der 160 Bit Message Digest.

Der SHA-1 Algorithmus kann durch folgende Gleichungen kompakt beschrieben werden:

$$\begin{aligned} \mathrm{CV}_0 &= \mathrm{IV} \\ \mathrm{CV}_{q+1} &= \mathrm{SUM}_{32}(\mathrm{CV}_q, \mathrm{ABCDE}_q) \\ \mathrm{MD} &= \mathrm{CV}_L \end{aligned}$$

wobei:

IV	=	Startwert des `ABCDE` Buffers, definiert in STEP 3.
ABCDE_q	=	der Output der letzten Verarbeitungsrunde des $q$ten Nachrichtenblocks
L	=	die Anzahl der 512-Bit Blöcke in der Nachricht
CV_q	=	Verkettungsvariable, die mit dem q-ten Nachrichtenblock verarbeitet wird
MD	=	der resultierende Message Digest Wert
SUM_{32}	=	Addition modulo 2^{32}

10.5.2 Die SHA-1 Kompressions Funktion

Sehen wir uns die Kompressionsfunktion des SHA-1 Algorithmus etwas näher an, wir untersuchen also, was in jeder der 80 Verarbeitungsrunden der 512-Bit Blöcke passiert. Jede Runde hat die Form:

$$\begin{aligned} \mathrm{A} &\leftarrow \mathrm{E} + f(t,\mathrm{B},\mathrm{C},\mathrm{D}) + S^5(\mathrm{A}) + \mathrm{W}_t + \mathrm{K}_t \\ \mathrm{B} &\leftarrow \mathrm{A} \\ \mathrm{C} &\leftarrow S^{30}(\mathrm{B}) \\ \mathrm{D} &\leftarrow \mathrm{C} \\ \mathrm{E} &\leftarrow \mathrm{D} \end{aligned}$$

dabei:

$\mathrm{A,B,C,D,E}$ = sind die fünf 32-Bit Worte des 160-Bit Puffers
t = Zähler der 80 Schritte $0 \leq t \leq 79$
$f(t,\mathrm{B,C,D})$ = bezeichnet die primitive logische Funktion
= im Schritt t
S^k = zirkulärer Left Shift des Arguments um k Bits
W_t = ein 32-Bit Wort, abgeleitet aus dem aktuellen
= 512-Bit Input Block
K_t = eine additive Konstante, oben definiert
$+$ = Addition modulo 2^{32}

Wie in der obigen Gleichung angedeutet, ist der Input jeder der primitiven logischen Funktion der Wert drei 32-Bit Register `B`, `C` und `D`, der Output ist ein 32-Bit Wort. Jede Funktion operiert bitweise auf folgende Weise:

$$\begin{aligned} f_1 &= f(t,\mathrm{A,B,C}) = (\mathrm{A} \wedge \mathrm{B}) \vee (\overline{\mathrm{A}} \wedge \mathrm{C}) \qquad 0 \leq t \leq 19 \\ f_2 &= f(t,\mathrm{A,B,C}) = \mathrm{A} \oplus \mathrm{B} \oplus \mathrm{C} \qquad 20 \leq t \leq 39 \\ f_3 &= f(t,\mathrm{A,B,C}) = (\mathrm{A} \wedge \mathrm{B}) \vee (\mathrm{A} \wedge \mathrm{C}) \vee (\mathrm{B} \wedge \mathrm{C}) \qquad 40 \leq t \leq 59 \\ f_4 &= f(t,\mathrm{A,B,C}) = \mathrm{A} \oplus \mathrm{B} \oplus \mathrm{C} \qquad 60 \leq t \leq 79 \end{aligned}$$

Hierbei bezeichnet $\wedge$ die BOOLEsche `AND` Operation, $\vee$ ist die `OR`-Operation, $\oplus$ bezeichnet die exclusive `OR` Operation (`XOR`) und $\overline{\ }$ ist die `NOT` Operation. Die Wahrheitstabelle dieser elementaren logischen Funktionen ist in der Tabelle [10.7] dargestellt.

A	**B**	**C**	f_1	f_2	f_3	f_4
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

Tabelle 10.7: Wahrheitstafel der primitven logischen Funktionen

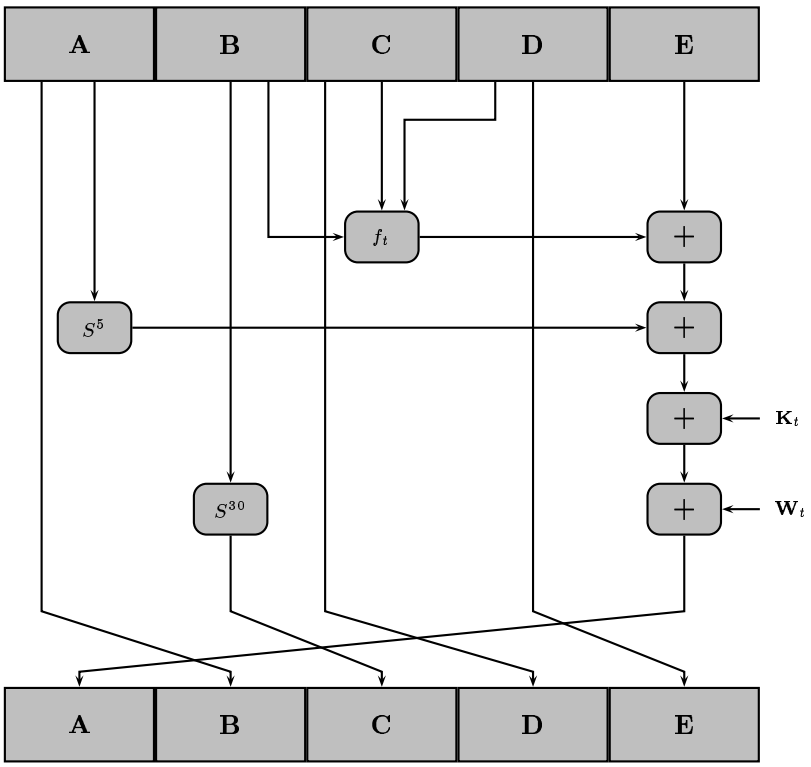


Abbildung 10.19: Grundlegende SHA-1 Operation

Wir müssen uns noch ansehen, wie die 32-Bit Worte W_t aus derm 512-Bit Nachrichtenblock abgeleitet werden. Die ersten 16 Werte von $textW_t$ werden direkt aus den 16 Worten des aktuellen Blocks genommen. Die restlichen Werte werden folgendermaßen abgeleitet:

$$\mathrm{W}_t = \mathrm{S}^1(\mathrm{W}_{t-16} \oplus \mathrm{W}_{t-14} \oplus \mathrm{W}_{t-8} \oplus \mathrm{W}_{t-3})$$

Also: In den ersten 16 Verarbeitungsschritten ist der Wert von W_t identisch mit dem entsrechenden Wort des aktuellen Nachrichtenblocks. Für die verbleibenden 64 Verarbeitungsschritte ist der Wert von W_t der Left Shift um eine Position der XOR-Verknüpfung von vier vorausberechneten Werten von W_t. Dies ist ein wesentlicher Unterschied zu MD5 und RIPEMD-160. Die beiden anderen Verfahren verwenden eins der 16 Worte eines Nachrichtenblocks direkt als Input in jedem Schritt, lediglich die Reihenfolge der Worte wird von RUnde zu Runde vertauscht. Im SHA-1 Algorithmus werden die 16-Block Worte auf 80 Worte für den Input in die Kopressionsfunktion erweitert. Die führt auf hohe Redundanzen und einen hohen Abhängigkeitgrad zwischen den Nachrichtenblöcken, die komprimiert werden. Dies erschwert beträchtlich die Absicht eines Angreifers, einen zweiten Nachrichtblock zu finden, der auf den gleichen Output der Kompressionsfunktion abgebildet wird.

10.5.3 Vergleich von SHA-1 und MD5

Da die beiden Hash Algorithmen MD5 und SHA-1 von RIVESTs MD4 Algorithmus abgeleitet sind, sind sie sich sehr ähnlich.

- **Sicherheit gegen Brute Force Attacken**
  Der offensichtliche und auch wesentliche Unterschied beider Hash Verfahren besteht in der Länge des Message Digests. Der Message Digest von SHA-1 hat eine Länge von 160 Bit, derjenige von MD5 128 Bit. Dementsprechend ist eine Brute-Force Attacke gegen MD5 von der Ordnung 2^{128} und ein Angriff gegen SHA-1 von der Ordnung 2^{160}. Mit Brute-Force Techniken resultiert, dass die Schwierigkeit, zwei Nachrichten zu erzeugen mit dem gleichen Message Digest, bei dem MD5 Algorithmus bei der Ordnung 2^{64} liegt, bei SHA-1 bei der Ordnung 2^{80}. Daraus folgt, dass SHA-1 wesentlcih stärker ist als MD5.

- **Sicherheit gegen Kryptoanalyse**
  Im Jahre 1996 wurde ein Angriff dokumentiert, der eine Kollision der MD5 Kompressionsfunktion erzeugte. Aus diesem Grund wird MD5 als angreifbar durch kryptoanalytische Angriffe angesehen. SHA-1 scheint resistent gegenüber solchen Angriffen zu sein (siehe jedoch den folgenden Abschnitt). Es ist jedoch wenig über die Design Kriterien des SHA-1 Verfahrens öffentlcih bekannt. Daher ist dessen Stärke schwer zu beurteilen.

- **Geschwindigkeit**
  Da beide Algorithmen auf Addition modulo 2^{32} basieren, haben beide Verfahren eine gute Performance auf 32 Bit Architekturen. SHA-1 beinhaltet mehr Verarbeitungsschritte als MD5 (80 gegenüber 64) und muß ein 160 Bit Puffer bearbeiten gegenüber einem 128 Bit Puffer des MD5 Algorithmus. Aus diesem Grund erwartet man, dass auf der gleichen Plattform SHA-1 etwas langsamer ist als MD5.

- **Einfachheit und Kompaktheit**
  Beide Algorithmen sind einfach zu beschreiben, einfach zu implemetieren und erfordern weder große Substitutionstabellen noch umfangreiche Programme.

10.5.4 Angriff auf SHA-1

Kapitel 11

Elliptische Kurven Kryptographie

11.1 Einführung

Mitte der 80er Jahre des letzten Jahrhunderts führten VICTOR S. MILLER [157] und NEAL KOBLITZ [128] elliptische Kurven in der Kryptographie ein[1]. HENDRIK LENSTRA entwickelte ein sehr elegantes und praktisch relevantes Verfahren ([141]), um mit Hilfe elliptischer Funktionen sehr große Zahlen in ihre Primfaktoren zu zerlegen [52, pp. 301–308]. Diese Entwicklungen führten dazu, dass Methoden der elliptischen Kurven immer mehr in die Kryptographie einflossen. Ein gewichtiger Vorteil, den die Verwendung elliptischer Kurven hat, ist, dass sie ein Sicherheitslevel bieten wie klassische Verschlüsselungsverfahren jedoch bei weitaus geringerer Schlüssellänge. So hat man gezeigt, dass konventielle Public Key Kryptosysteme mit einem 4096 Bit Schlüssel durch ein entsprechendes Verfahren mit elliptischen Kurven ersetzt werden kann, das nur eine Schlüssellänge von 313 Bits hat bei gleicher Sicherheit (siehe dazu die Diskussion in [25], Chapter I.3). Die Verwendung kleinerer Zahlen bietet große Vorteile bei Hardwareimplementierungen von kryptographischen Systemen.

Die herkömmlichen Public-Key Systeme wie RSA, Diffie-Hellman oder ElGamal bilden Trapdoro Funktionen auf multiplikativen Gruppen wie $(\mathbb{Z}_n^*, \times$ oder $(\mathbb{Z}_p^*, \times)$, p Primzahl. Wie in [131] dargestellt ist, war eine der Motivation von KOBLITZ und MILLER die Untersuchung der Möglichkeit, ob es weitere Gruppen gibt, die One-Way Funktionen ermöglichen. Es hat sich herausgestellt, dass elliptischen Kurven sehr viele Gruppen ermöglichen – also nicht nur eine wie in den traditionellen Public-Key Systemen – über denen Kryptosysteme aufgebaut werden können.

In diesem Kapitel sehen wir uns einige Aspekte der elliptischen Kurven Kryptographie an. Zunächst werden elliptische Kurven über den reellen Zahlen betrachtet. Dies ist zunächst für die Kryptographie nicht relevant, bietet jedoch den Vorteil einer gewissen geometrischen Anschaulichkeit der durchgeführten

[1] Siehe den Artikel von NEAL KOBLITZ [131], der die Entstehung der Kryptosysteme auf elliptischen Kurven aus Sicht der Entwickler schildert.

Konstruktionen. Die hier über den reellen Zahlen diskutierten Verfahren können direkt ohne große Änderungen auf endliche Körper (e.g. $\mathbb{Z}_p$) übertragen werden.

Es gibt eine Reihe von Einführungen in dieses Themengebiet, siehe z.B. [215, pp. 272–290], [227], [218] [213]. Einen sehr guten Überblick über den algebraischen Hintergrund der Public Key Kryptographie auf elliptischen Kurven findet man in dem Artikel von HENRI COHEN [46]. Empfehlenswert ist die Monographie von LAWRENCE WASHINGTON [222].

Die meisten Produkte und Standards, die Public-Key Kryptosysteme zur Ver-/Entschlüsselung und Digitale Signaturen einsetzen, basieren auf dem RSA Algorithmus. Über die letzen Jahre ist jedoch die Schlüssellänge des RSA Verfahrens immer weiter gewachsen, um die Sicherheit des Kryptosystems zu gewährleisten. Dies führt natürlich auf eine höhere Rechenzeitbelastung auf die Anwendungen, die RSA nutzen. Die elliptischen Kurven Kryptosysteme (ECC) finden aufgrund dessen immer mehr an Verbreitung. Für elliptische Kurven Kryptosysteme sind bereits eine Reihe von Standards veröffentlicht:

- ☞ IEEE P1363 Standard für Public Key Kryptographie.
- ☞ NIST Standard FIPS 186-2. Diese Spezifikation des DSS (Digital Signature Standard beinhaltet eine Standardisierung des *elliptic curve digital signature algorithm* (ECDSA) mit Empfehlungen für 15 Parametersets für elliptische Kurven.
- ☞ ANSI X9; wie in Abschnitt [1.3] erwähnt, ist die Aufgabe des ANSI X9 Gremiums die Entwicklung von Standards in der Finanzdienstleistungsindustrie; die folgenden Standards beziehen sich auf elliptische Kurven Kryptosysteme:
  - American National Standard X9.62: *The Elliptic Curve Digital Signature Algorithm*
  - American National Standard X9.63: *Key Agreement and Key Transport Using Elliptic Curve Cryptography*

Die kanadische Firma Certicom Inc.[2] ist im Besitz sehr vieler Patente über elliptische Kurven Kryptographie und Public Key Kryptographie im Allgemeinen. Dies hemmt die kommerzielle Verbreitung dieser Kryptosysteme.

[2] Siehe die Website [261]. Betrachtet man die Liste der Mitarbeiter von Certicom, findet man ein *who-is-who* der elliptische Kurven Kryptographie.

Symm. Key (Bit)	RSA / DH	ECC
56	512	112
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Tabelle 11.1: Vergleichbare Schlüsselgrößen bei symmetrischen Kryptosystemen, klassischen Public-Key Verfahren wie RSA und DIFFIE-HELLMAN sowie elliptische Kurven Kryptosystemen. Die Schlüsselgröße in Bit ist ein Maß für die Sicherheit, die die Systeme bieten, gleichzeitig ein Maß für den Rechenaufwand in der Kryptoanalyse, um ein System zu brechen. (Quelle: [261])

11.2 Elliptische Kurven über den reellen Zahlen

Ausgangspunkt unserer Betrachtungen sind die sogenannten **Diophantische Gleichungen** (siehe z.B. [8]) der Form

$$y^2 = x^3 + ax + b \tag{11.1}$$

wobei die Koeffizienten a, b Elemente eines Körpers sind, i.e. aus den reellen oder komplexen Zahlen oder aus einem endlichen Körper. Gesucht sind nun *ganzzahlige* Lösungen solcher Gleichungen. Ziel ist es, ein Verfahren zu konstruieren, das aus zwei bekannten Lösungen – die auch identisch sein können – eine dritte Lösung der diophantischen Gleichung liefert. Mit anderen Worten, gesucht ist eine Operation auf der Menge der ganzzahligen Lösungen einer elliptischen Kurve, so daß die Menge der ganzzahligen Lösungen abgeschlossen ist unter dieser Operation.

Beispiel:

Das folgende Beispiel aus [222] zeigt eine der typischen Fragestellungen, die mit Hilfe elliptischer Kurven diskutiert werden können.

Abbildung 11.1: Eine Pyramide aus Bällen.

Angenommen eine Reihe von Bällen (oder Kugeln) sind in Form einer quadratischen Pyramide – wie in der Abbildung [11.1] dargestellt – aufgestapelt. Ein Ball liegt ganz oben, die zweite Schicht besteht aus vier Kugeln, die dritte aus neun usw. Die Frage ist nun: Falls nun die Pyramide zusammenbricht, ist es möglich, die Kugeln in Form eines (ebenen) Quadrates anzuordnen?

Falls die Pyramide aus drei Schichten besteht, ist es nicht möglich, die Bälle in Form eines ebenen Quadrates anzuordnen, denn bei drei Schichten hat man

$$1 + 4 + 9 = 14$$

Bälle, was keine Quadratzahl ist. Wenn man eine Pyramide mit nur einem Ball hat, dann bildet dies zugleich ein 1 - 1 Quadrat. Falls die Pyramide aus keinem Ball besteht, haben wir ein 0 - 0 Quadrat. Die Frage, die sich bereits der Grieche DIOPHANTUS um 250 n. Chr. gestellt hat, ist nun, ob es neben diesen trivialen Fällen weitere Anordnungen gibt, so daß die Pyramide in ein Quadrat umgewandelt werden kann.

Falls die Pyramide aus x Schichten besteht, dann benötigt man zu Bau dieser Pyramide

$$1^2 + 2^2 + 3^2 + \ldots + x^2 = \frac{x(x+1)(2x+1)}{6}$$

Bälle. Dieser Ausdruck muß ein Quadrat sein, i.e., gesucht sind positive ganzzahlige Lösungen (x, y) der Gleichung

$$\begin{aligned} y^2 &= \frac{x(x+1)(2x+1)}{6} \\ &= \frac{1}{3}x^3 + \frac{1}{2}x^2 + \frac{1}{6}x \end{aligned}$$

Eine Gleichung dieser Form stellt eine **elliptische Kurve** dar[3]. Der Graph dieser elliptischen Funktion ist in der Abbildung [11.2] dargestellt.

Die Methode von DIOPHANTUS benutzt nun die beiden bereits bekannten Punkte auf der elliptischen Kurve, um einen neuen Punkt auf dieser Kurve zu erhalten. Wir starten dehr mit den beiden Punkten

$$P_0 = (0, 0); P_1 = (1, 1)$$

Die Gerade durch diese beiden Punkte ist einfach

$$y = x$$

Wenn wir die Schnittpunkte dieser Geraden mit der gegebenen elliptischen Kurve bestimmen wollen, müssen wir diese Geradengleichung in die Gleichung der elliptischen Kurve einsetzen. Es folgt:

$$x^2 = \frac{1}{3}x^3 + \frac{1}{2}x^2 + \frac{1}{6}x$$

[3] Die Form dieser Gleichung unterscheidet sich von der Gleichung [11.1] dahingehend, daß hier ein quadratischer Term auftritt. Durch eine geeignete Variablensubstitution kann dieser quadratische Term jedoch stets wegtransformiert werden, so daß eine Gleichung der Form [11.1] resultiert.

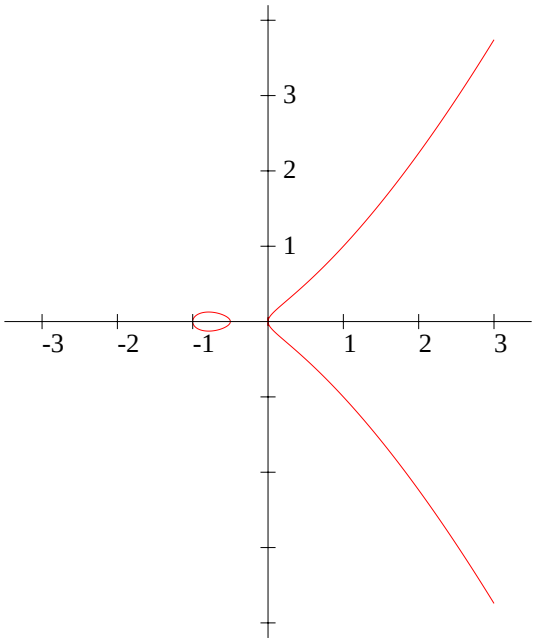


Abbildung 11.2: Der Graph der elliptischen Kurve $y^2 = \frac{1}{3}x^3 + \frac{1}{2}x^2 + \frac{1}{6}x$.

oder

$$x^3 - \frac{3}{2}x^2 + \frac{1}{2}x = 0$$

Ziel ist es nun, alle Nullstellen dieser Gleichung zu finden, diese heißen auch Wurzeln. Die Wurzeln dieser Gleichung sind die x-Koordinaten der Schnittpunkte der Geraden $y = x$ mit der elliptischen Kurve.

Nun ist:

$$\begin{aligned} x^3 - \frac{3}{2}x^2 + \frac{1}{2}x &= x\left(x^2 - \frac{3}{2}x + \frac{1}{2}\right) \\ &= 0 \end{aligned}$$

mit der bereits bekannten Lösung

$$x_1 = 0$$

Die beiden anderen Wurzeln sind daher Lösungen der quadratischen Gleichung

$$x^2 - \frac{3}{2}x + \frac{1}{2} = 0$$

oder

$$x_{2/3} = \frac{3}{4} \pm \frac{1}{2}$$

Daher

$$x_2 = 1; x_3 = \frac{1}{2}$$

woraus also der dritte Schnittpunkt

$$P_3 = (\frac{1}{2}, \frac{1}{2})$$

resultiert.

Anmerkung:

Neben diesem Standardverfahren der Berechnung der Nullstellen eines Polynoms dritter Ordnung kann die dritte Wurzel auch folgendermaßen abgeleitet werden, wenn die beiden anderen Wurzeln bereits bekannt sind. Ein Polynom dritten Grades mit Wurzeln a, b, c kann immer in der Form

$$(x-a)(x-b)(x-c) = x^3 + (a+b+c)x^2 + (ab+ac+bc)x - abc$$

geschrieben werden. Mit anderen Worten, wenn der Koeffizient des x^3-Terms 1 ist, ist der ist das Negative des Koeffizienten von x^2 die Summe der drei Wurzeln. Damit erhalten wir einfach: (mit a=0, b=1)

$$0 + 1 + c = \frac{3}{2}$$

oder

$$c = \frac{1}{2}.$$

Formal haben wir also eine Lösung unseres ursprünglichen Problems erhalten, i. e. den Punkt $P_3 = (\frac{1}{2}, \frac{1}{2})$ abgeleitet. Es ist etwas schwierig, dieses Ergebnis durch die Bälle zu interpretieren.

Aufgrund der Symmetrie der elliptischen Kurve – i.e. die Spiegelsymmetrie zur y Achse, haben wir sogar einen weiteren Punkt abgeleitet, der ebenfalls auf der Kurve liegt, nämlich der Punkt

$$P_4 = (\frac{1}{2}, -\frac{1}{2}).$$

Wir wiederholen das obige Verfahren mit den beiden Punkten $P_0 = (\frac{1}{2}, -\frac{1}{2})$ und $P_1 = (1, 1)$. Wir wählen diese beiden Punkte aus dem Grund, da wir einen Schnittpunkt im ersten Quadranten des Koordinatensystems suchen[4]. Die Gerade durch die beiden Punkte ist

$$y = 3x - 2$$

Einsetzen dieser Geradengleichung in die Gleichung der elliptischen Kurve ergibt

$$(3x-2)^2 = \frac{x(x+1)(2x+1)}{6}$$

[4]Die Anzahl von Bällen sowohl in der Pyramide als auch im Quadrat ist nun mal positiv.

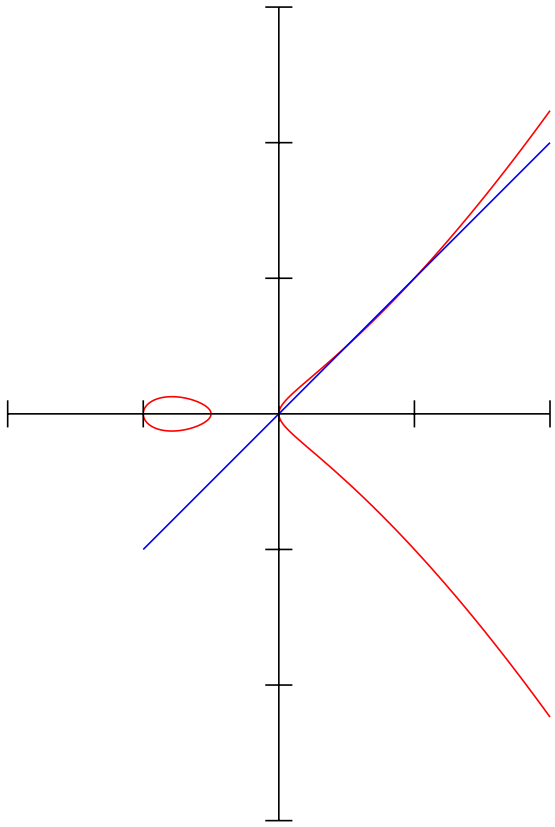


Abbildung 11.3: Die elliptische Kurve $y^2 = \frac{1}{3}x^3 + \frac{1}{2}x^2 + \frac{1}{6}x$ mit der Geraden $f(x) = x$.

oder

$$x^3 - \frac{51}{2}x^2 + \frac{73}{2}x - 12 = 0$$

Die beiden Wurzeln $a = \frac{1}{2}$ und $b = 1$ sind bereits bekannt, damit erhält man die dritte Wurzel zu:

$$\frac{1}{2} + 1 + c = \frac{51}{2},$$

oder $c = 24$. Aus der Geradengleichung $y = 3x - 2$ folgt damit $y = 70$. Das Ergebnis hat also die Bedeutung:

$$1^2 + 2^2 + \ldots + 24^2 = 70^2$$

oder: Falls man 4900 Bälle zur Verfügung hat, können diese in Form einer quadratischen Pyramide mit 24 Schichten oder als 70 mal 70 Quadrat arrangiert werden.

Beispiel:

Wir diskutieren den Fall einer reellen elliptischen Funktion mit den Parametern $b = 0, a = -2$, i.e. die diophantische Gleichung

$$y^2 = x^3 - 2x$$

Diskussion:

① **Definitionsbereich**
Da wir ausschließlich im Reellen arbeiten, darf die linke Seite der Gleichung nicht negativ werden. Dies ist dann der Fall, wenn

$$x(x^2 - 2) \geq 0$$

oder $x \geq 0$ und $x^2 \geq 2$. Damit sind die zulässigen x Werte

$$\mathcal{D} = \{x \in \mathbb{R} \mid -\sqrt{2} \leq x \leq 0 \text{ oder } x \geq +\sqrt{2}\}$$

② **Symmetrieeigenschaften**
Da mit dem Wertepaar (x_1, y_1) auch $(x_1, -y_1)$ eine Lösung der Gleichung ist, ist der Graph der elliptischen Kurve spiegelsymmetrisch zur x Achse.

③ **Nullstellen**
Die Nullstellen sind definiert durch $y = 0$ oder

$$0 = x(x^2 - 2) = x(x + \sqrt{2})(x - \sqrt{2})$$

Das ist genau dann der Fall bei

$$\begin{aligned} x_1 &= 0 \\ x_2 &= -\sqrt{2} \\ x_3 &= +\sqrt{2} \end{aligned}$$

④ **Ableitungen:**
Differenziert man implizit die Gleichung der elliptischen Kurve, erhält man:

$$2yy' = 3x^2 - 2$$

oder

$$\frac{dy}{dx} = \pm\frac{3x^2 - 2}{2\sqrt{x^3 - 2x}}$$

Die Nullstellen der Ableitung sind bei

$$x_{1/2} = \pm\sqrt{\frac{2}{3}}$$

Daher ist an der Stelle

$$x = \sqrt{\frac{2}{3}}$$

ein lokales Extremum (ohne Beweis Maximum). Bei den Nullstellen des Nenners der Ableitungen treten vertikale Tangenten auf, das sind genau die Nullstellen der elliptischen Funktion.

Der Graph der elliptischen Funktion

$$y^2 = x^3 - 2x$$

ist in der Abbildung [11.4] dargestellt.

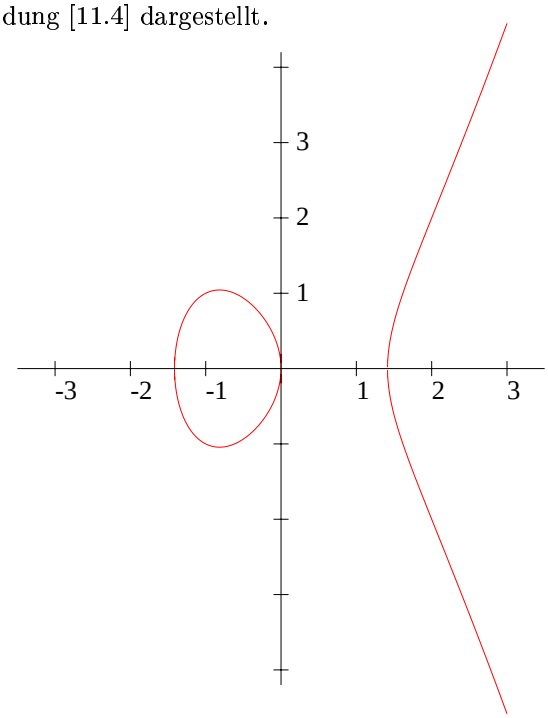


Abbildung 11.4: Graph der elliptischen Funktion $y^2 = x^3 - 2x$.

Gesucht sind nun ganzzahlige Lösungen der Gleichung,

$$y^2 - x^3 + 2x = 0$$

i.e. die Lösungsmenge

$$\mathbb{L} = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \text{ mit } y^2 - x^3 + 2x = 0\}$$

Offensichtlich gilt:

- ✘ $(x_1, y_1) = (0, 0)$ ist eine ganzzahlige Lösung.
- ✘ $(x_2, y_2) = (1, -1)$ ist eine ganzzahlige Lösung
- ✘ $(x_3, y_3) = (-1, -1)$ ist eine ganzzahlige Lösung

Wir konstruieren nun ein Verfahren, das aus zwei bekannten ganzzahligen Lösungen der elliptischen Funktion eine neue Lösung generiert.

Wenn zwei Punkte (x_1, y_1) und (x_2, y_2) in der Ebene gegeben sind, dann ist die Gleichung der Geraden, die durch diese beiden Punkte geht:

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1) + y_1 = m \cdot (x - x_1) + y_1$$

mit der Steigung

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Check:

Die allgemeine Geradengleichung ist

$$y = mx + b$$

Wir haben die beiden Bedingungen:

$$y_1 = mx_1 + b$$
$$y_2 = mx_2 + b$$

da diese beiden Punkte ja auf der Gerade liegen. Diese beiden Bedingungen sind nun nach den beiden Unbekannten m und b aufzulösen. Subtrahiert man die erste Gleichung von der zweiten folgt:

$$y_2 - y_1 = m \cdot (x_2 - x_1)$$

oder

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Einsetzen dieses Werts in die erste Gleichung liefert:

$$y_1 = \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1 + b$$

oder

$$b = y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1$$

Damit resultiert in der Tat:

$$\begin{aligned} y &= m \cdot x + b \\ &= \frac{y_2 - y_1}{x_2 - x_1} \cdot x + y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1 \\ &= \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1) + y_1 \end{aligned}$$

Wir stellen nun die Gleichung der Geraden auf, die durch die beiden Punkte

$$P = (x_1, y_1) = (-1, 1)$$
$$Q = (x_2, y_2) = (0, 0)$$

geht, die ganzzahlige Lösungen der elliptischen Kurve sind. Es folgt durch Einsetzen in die Geradengleichung:

$$\begin{aligned} y &= \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1) + y_1 \\ &= \frac{0 - 1}{0 + 1} \cdot (x + 1) + 1 \\ &= -x \end{aligned}$$

Damit erhalten wir das (offensichtliche) Ergebnis, daß die Gerade

$$y = -x$$

durch die beiden Punkte $(-1, 1)$ und $(0, 0)$ geht.

Um nun einen dritten Punkt zu erhalten, der

① auf der Gerade liegt

② auf der elliptischen Kurve liegt,

setzt man die gefundene Geradengleichung in die Gleichung für die elliptische Kurve ein. Damit:

$$\begin{aligned} 0 &= y^2 - x^3 + 2x \\ &= x^2 - x^3 + 2x \\ &= x \cdot (x - x^2 + 2) \\ &= -x \cdot (x^2 - x - 2) \\ &= x \cdot (x + 1) \cdot (x - 2) \end{aligned}$$

mit den drei Lösungen $x = 0, x = -1$ und $x = 2$. Dies zeigt, daß die Gerade drei (ganzzahlige) Schnittpunkte mit der elliptischen Kurve hat:

$$\begin{aligned} P &= (x_1, y_1) = (-1, 1) \\ Q &= (x_2, y_2) = (0, 0) \\ R &= (x_3, y_3) = (2, 2) \end{aligned}$$

Dieses Verfahren wollen wir nun allgemein betrachten.

Definition

Seien $a, b \in \mathbb{R}$ mit $a, b = const.$ mit

$$4a^3 + 27b^2 \neq 0$$

Eine nicht-singuläre, elliptische Kurve ist die Menge der Lösungen $(x, y) \in \mathbb{R} \times \mathbb{R}$ der Gleichung

$$y^2 = x^3 + ax + b$$

zusammen mit dem speziellen Punkt $P = \infty$ im Unendlichen.

Anmerkungen:

① Man kann zeigen, daß die Bedingung

$$4a^3 + 27b^2 \neq 0$$

notwendig und hinreichend für die Existenz dreier reeller Wurzeln ist. Dies ist analog zu der Diskriminante bei der Untersuchung der Menge der Lösungen einer quadratischen Gleichung

$$ax^2 + bx + c = 0$$

mit

$$x_{1/2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Es existieren genau dann zwei reelle Lösungen, wenn für die Diskriminante gilt:

$$b^2 - 4ac > 0$$

② Falls

$$4a^3 + 27b^2 = 0$$

dann heißt die elliptische Kurve **singulär**. Eine singuläre elliptische Kurve ist zum Beispiel:

$$y^2 = x^3 - 3x + 2 = (x-1)^2(x+2)$$

Diese Kurve hat einen Doppelpunkt im Punkt $P = (1, 0)$. Der Graph dieser Funktion ist in der Abbildung [11.5] dargestellt.

Eine andere singuläre elliptische Kurve ergibt sich dann, wenn die beiden Parameter a und b beide verschwinden. Die elliptische Kurve hat dann im Nullpunkt einen Rückkehrpunkt.

$$y^2 = x^3$$

Der Graph dieser Kurve ist in der Abbildung [11.6] dargestellt.

③ Die allgemeinste Form einer elliptischen Kurve über einem Körper K mit Charakteristik > 3 ist (siehe [107, Chapt. 3]):

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad a_i \in K$$

Diese Form der elliptischen Gleichung nennt man WEIERSTRASS Gleichung. Durch die folgende Transformation der Variablen x, y kann die WEIERSTRASS Gleichung der elliptischen Kurve auf die Form

$$E : y^2 = x^3 + ax + b \quad a, b \in K$$

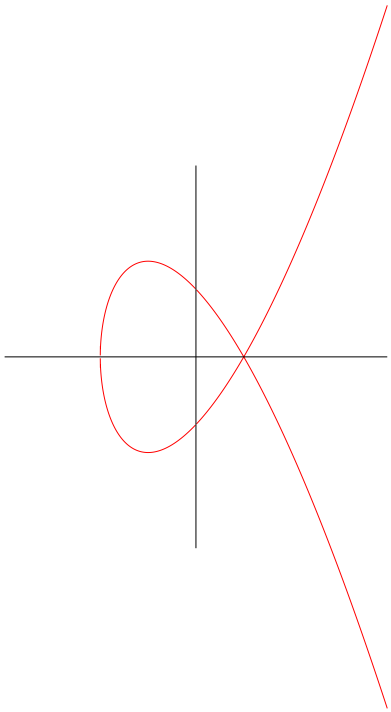


Abbildung 11.5: Die elliptische Funktion $y^2 = x^3 - 3x - 2$

.

gebracht werden:

$$x \longrightarrow x' = \frac{x - 3a_1^2 - 12a_2}{36}$$
$$y \longrightarrow y' = \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24}$$

Beweis:

---

④ Elliptische Kurven haben nichts mit Ellipsen zu tun. Der Terminus *elliptische Kurve* stammt aus der Untersuchung der Bogenlängen von Ellipsen. Dabei treten folgende Integrale auf:

$$\int \frac{dx}{\sqrt{x^3 + ax + b}} \text{ bzw. } \int \frac{x\,dx}{\sqrt{x^3 + ax + b}}$$

Das Argument der Wurzel im Nenner hat die gleiche algebraische Form wie die elliptischen Kurven.

⑤ Arbeitet man in Körpern mit Charakteristik > 3, ist es ausreichend, die Form

$$y^2 = x^3 + ax + b$$

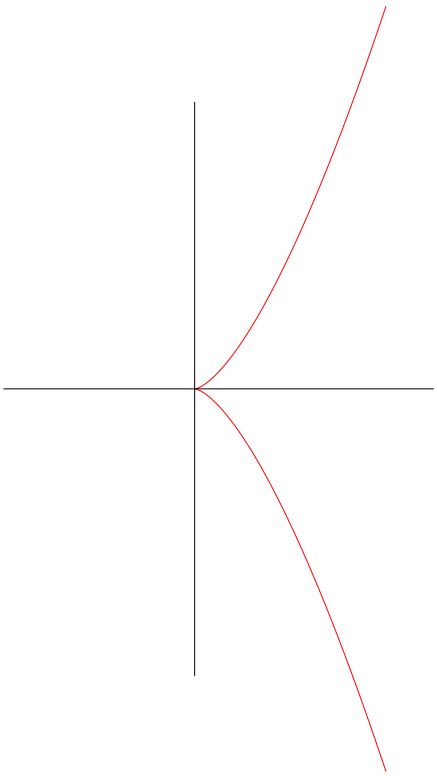


Abbildung 11.6: Die elliptische Kurve $y^2 = x^3$.

als Gleichung für elliptische Kurven zu betrachten.

Angenommen E ist eine elliptische Kurve über dem Körper der reellen Zahlen. Ziel ist es nun, eine binäre Operation auf E zu konstruieren, so daß auf E eine Gruppenstruktur induziert wird. Der Grund, warum wir hier eine auf elliptischen Kurven definierte Gruppenstruktur konstruieren liegt darin, daß ElGamal Kryptosysteme auf allen Gruppen implementiert werden können, auf denen das *diskrete Logarithmusproblem* praktisch unlösbar ist. Dabei werden drei Arten betrachtet:

❶ ElGamal Kryptosysteme auf der multiplikativen Gruppe $(\mathbb{Z}_p^*, \cdot)$.

❷ ElGamal Kryptosysteme auf der multiplikativen Gruppe des endlichen Körpers $\mathbb{F}_{p^n}$.

❸ ElGamal Kryptosysteme auf der Gruppe einer elliptischen Kurve über einem endlichen Körper.

Der Punkt ∞ ist das neutrale Element der Operation, die üblicherweise mit $'+'$ bezeichnet wird, da die Gruppe additiv ist. Damit:

$$P + \infty = \infty + P = P \quad \forall\, P \in E$$

Wir nehmen nun an, daß $P, Q \in E$ zwei Punkte der elliptischen Kurve E gegeben sind mit:

$$P = (x_1, y_1)$$
$$Q = (x_2, y_2)$$

Bei der Konstruktion der binären Operation

$$+ \; : \; E \times E \longrightarrow E$$

unterscheidet man drei Fälle:

❶ $x_1 \neq x_2$

❷ $x_1 = x_2$ und $y_1 = -y_2$

❸ $x_1 = x_2$ und $y_1 = y_2$

Fall 1: **Konstruktionsidee:**
Wir definieren L als die Gerade, die durch die beiden Punkte P und Q geht. Die Gerade L schneidet daher die elliptische Kurve E in den beiden Punkten P und Q. Da die elliptische Kurve nicht-singulär ist, schneidet diese Gerade die elliptische Kurve in einem dritten Punkt R'. Spiegelt man diesen Punkt an der x Achse erhält man einen dritten Punkt $R \in E$. Wir setzen:

$$P + Q = R$$

Dieses Konstruktionsprinzip ist in der Abbildung [11.7] skizziert.

Details der Konstruktion

Ziel ist es, einen algebraischen Ausdruck für die Koordinaten des Punktes R zu finden, wenn die beiden Punkte $P, Q \in E$ vorgegeben sind.

Sei also

$$P = (x_1, y_1)$$
$$Q = (x_2, y_2)$$

gegeben. Die Geradengleichung für L durch die beiden Punkte P und Q ist:

$$y = \lambda x + \nu$$

mit der Steigung

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

und dem Achsenabschnitt

$$\nu = y_1 - \lambda x_1 = y_2 - \lambda x_2$$

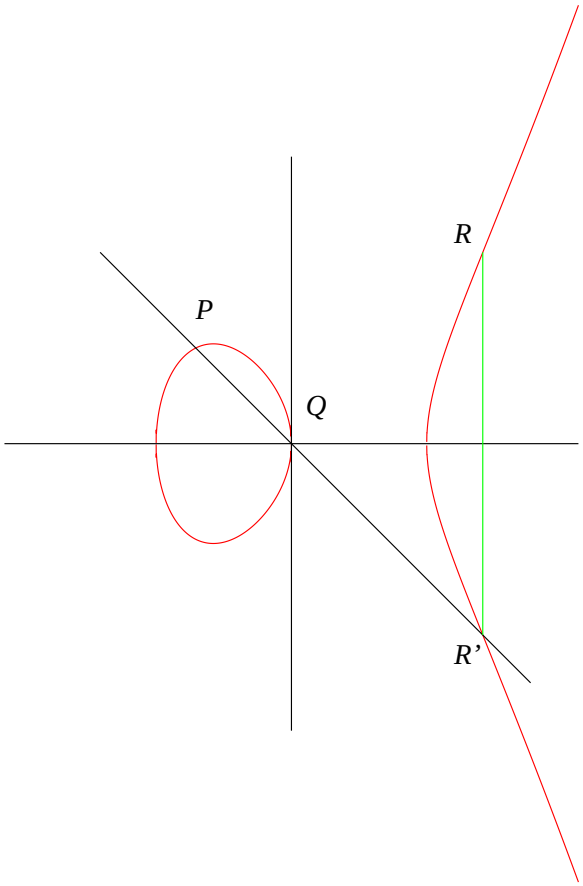


Abbildung 11.7: Konstruktion des dritten Punktes auf der elliptischen Kurve.

Check:

$$
\begin{aligned}
y_1 - \lambda x_1 &= y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1 \\
&= y_1 - \frac{y_2 x_1}{x_2 - x_1} + \frac{y_1 x_1}{x_2 - x_1} \\
&= -\frac{y_2 x_1}{x_2 - x_1} + y_1 (1 + \frac{x_1}{x_2 - x_1}) \\
&= -\frac{y_2 x_1}{x_2 - x_1} + y_1 (\frac{x_2}{x_2 - x_1}) \\
&= -\frac{y_2 x_1}{x_2 - x_1} + \frac{y_1 x_2}{x_2 - x_1} \\
&= y_2 - y_2 - \frac{y_2 x_1}{x_2 - x_1} + \frac{y_1 x_2}{x_2 - x_1} \\
&= y_2 - \frac{y_2 (x_2 - x_1)}{x_2 - x_1} - \frac{y_2 x_1}{x_2 - x_1} + \frac{y_1 x_2}{x_2 - x_1} \\
&= y_2 - \frac{y_2 \cdot x_2}{x_2 - x_1} + \frac{y_1 x_2}{x_2 - x_1} \\
&= y_2 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_2 \\
&= y_2 - \lambda \cdot x_2
\end{aligned}
$$

Um die Punkte $E \cap L$ – i.e. alle Schnittpunkte der Geraden L mit der elliptischen Kurve E – zu finden, setzt man die Geradengleichung in die Gleichung für die elliptische Kurve ein.

$$
\begin{aligned}
y^2 &= x^3 + ax + b \\
\iff (\lambda x + \nu)^2 &= x^3 + ax + b \\
\iff \lambda^2 x^2 + 2\lambda\nu x + \nu^2 &= x^3 + ax + b
\end{aligned}
$$

oder

$$x^3 - \lambda^2 x^2 - (a - 2\lambda)x + b - \nu^2 = 0$$

Die Wurzeln dieser Gleichung – i.e. diejenigen x Werte, die diese Gleichung lösen – sind die x Koordinaten der Punkte in der Schnittmenge $E \cap L$.

Zwei Punkte in $E \cap L$ sind bereits bekannt, das sind die beiden Punkte P und Q. Daher sind x_1 und x_2 zwei bekannte Wurzeln dieser Gleichung.

Da die Gleichung

$$x^3 - \lambda^2 x^2 - (a - 2\lambda)x + b - \nu^2 = 0$$

eine kubische Gleichung über $\mathbb{R}$ ist mit zwei reellen Lösungen x_1, x_2, muß die dritte Wurzel – diese nennen wir x_3 – ebenfalls reell sein.

Behauptung:

> Die Summe der drei Wurzeln einer kubischen Gleichung ist das Negative des Koeffizienten des quadratischen Terms.

Check:

Die allgemeine kubische Gleichung ist

$$x^3 + ax^2 + bx + c = 0$$

mit drei reellen Lösungen x_1, x_2, x_3. Dann folgt:

$$\begin{aligned}
0 &= (x - x_1) \cdot (x - x_2) \cdot (x - x_3) \\
&= (x^2 - xx_2 - xx_1 + x_1x_2) \cdot (x - x_3) \\
&= x^3 - x^2x_2 - x^2x_1 + x_1x_2x - x^2x_3 + xx_2x_3 + xx_1x_3 - x_1x_2x_3 \\
&= x^3 - x^2(x_1 + x_2 + x_3) + x(x_1x_2 + x_2x_3 + x_1x_3) - x_1x_2x_3 \\
&\stackrel{!}{=} x^3 + ax^2 + bx + c
\end{aligned}$$

Dies ist genau dann der Fall, wenn

$$a = -(x_1 + x_2 + x_3)$$

gilt.

Mit dieser Eigenschaft kubischer Polynome finden wir:

$$x_3 = \lambda^2 - x_2 - x_1$$

x_3 ist damit die x-Koordinate des gesuchten Punktes R'. Wir benennen die y-Koordinaten von R' mit $-y_3$, damit hat der gesuchte Punkte R – das ist der zu R' an der x Achse gespiegelte Punkt – die Koordinate y_3.

Um y_3 auf einfachste Weise festzulegen, benutzen wir die Tatsache, daß die Steigung einer Gerade nicht nur durch die beiden Punkte P und Q festgelegt ist, sondern durch zwei *beliebige* Punkte, die auf der Geraden L liegen. Aus diesem Grund kann man für die Steigung λ auch die beiden Punkte P und R' verwenden, i.e. wir benutzen die Punkte

$$P = (x_1, y_1) \text{ und } R' = (x_3, -y_3)$$

Damit wird

$$\lambda = \frac{-y_3 - y_1}{x_3 - x_1}$$

oder

$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

Damit haben wir den Fall 1 abgeschlossen:

Falls $P, Q \in E$, E ist elliptische Kurve und

$$P = (x_1, y_1), Q = (x_2, y_2)$$

und $P \neq Q$, dann ist der Punkt

$$R = P + Q \in E$$

gegeben durch

$$P + Q = (x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

mit den Koordinaten

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

und der Steigung:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

Fall 2: **Konstruktionsidee:**
Der zweite Fall ist $x = x_1 = x_2$ und $y = y_1 = -y_2$. Das heißt, die beiden Punkte haben den gleichen x Wert aber das Vorzeichen des y Wertes unterscheidet sich. Dieser Fall ist in der Abbildung [11.8] dargestellt. Diesen Fall definieren wir durch:

$$(x, y) + (x, -y) = \infty \quad \forall\, (x, y) \in E$$

Das bedeutet, die Punkte $P = (x, y)$ und $Q = (x, -y)$ sind invers bezüglich der binären Operation über E.

Fall 3: **Konstruktionsidee:**
Bei diesem entarteten Fall fallen die beiden Punkte P und Q zusammen, i.e.

$$P = (x_1, y_1) = Q = (x, y)$$

Hier ist also die Operation

$$2P = P + P = R$$

zu berechnen. Wir nehmen o.B.d.A. an, daß $y \neq 0$, sonst reduziert sich dies auf den Fall 2. Dann ist dieser Fall sehr ähnlich dem in 1, mit der Ausnahme, daß anstatt einer Sekante eine Tangente an die Kurve

$$E: \; y^2 = x^3 + ax + b$$

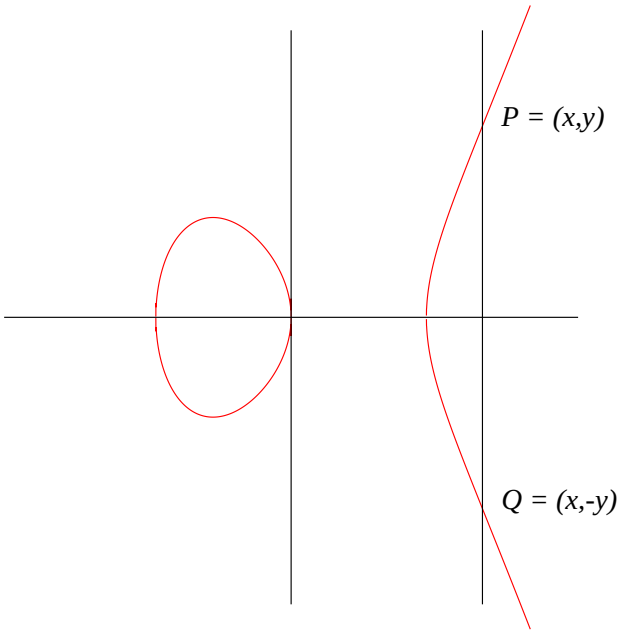


Abbildung 11.8: Konstruktion des dritten Punktes auf der elliptischen Kurve für den Fall $P = -Q$.

im Punkt $P = (x, y)$ zu untersuchen ist. Diese Situation ist in Abbildung [11.9] dargestellt.

Um die Steigung der Tangente im Punkt $P = (x_1, y_1)$ zu finden, differenziert man die Gleichung der elliptischen Kurve implizit, i.e.

$$2y\frac{dy}{dx} = 3x^2 + a$$

oder

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$

Der Rest der Analyse ist wie im Fall 1 mit Ausnahme der expliziten Form der Steigung.

Damit: Es gilt

$$P + P = 2P = R$$

mit

$$P = (x_1, y_1) \text{ und } R = (x_3, y_3)$$

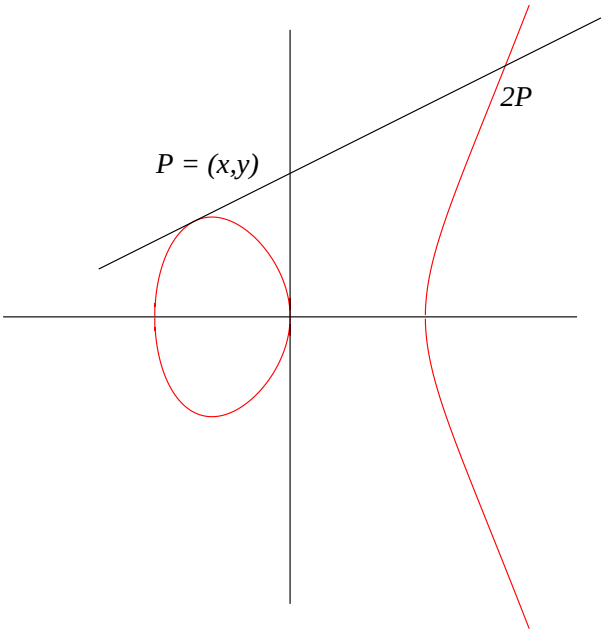


Abbildung 11.9: Konstruktion des dritten Punktes auf der elliptischen Kurve für den Fall $P = Q$.

sowie

$$x_3 = \lambda^2 - 2x_1$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

und der Steigung

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$

An dieser Stelle sind die folgenden Eigenschaften der Additionsoperation – wie oben definiert – Fakt:

① Die Menge E ist abgeschlossen unter der Additionsoperation.

② Die Additionsoperation ist kommutativ, i.e.

$$P + Q = Q + P$$

③ ∞ ist das neutrale Element der Additionsoperation.

④ Jeder Punkt $P = (x, y) \in E$ hat einen inversen Punkt $P' = (x, -y) \in E$ mit

$$P + P' = \infty$$

⑤ Was noch gezeigt werden muß ist, daß die definierte Addition das Assoziativgesetz erfüllt, i.e.

$$(P + Q) + R = P + (Q + R)$$

Dies ist sehr lästig.

Die obigen Eigenschaften der elliptischen Kurven E mit der definierten Additionsoperation '+' implizieren, daß $(E, +)$ die Struktur einer ABELschen Gruppe hat.

11.2.1 Sukzessives Verdoppeln

Falls P ein Punkt auf einer elliptischen Kurve bezeichnet und k eine positive ganze Zahl, dann bezeichnet

$$k \cdot P = \underbrace{P + P + \cdots + P}_{k \text{ mal}}.$$

Ist $k < 0$, dann ist

$$k \cdot P = \underbrace{(-P) + (-P) + \cdots + (-P)}_{|k| \text{ mal}}.$$

Um kP zu berechnen für große Werte von k ist es ineffizient, P einfach wiederholt mit sich selbst zu addieren. Eine sehr effiziente Methode dies zu tun, ist das **sukzessive Verdoppeln**. Um beispielsweise den Punkt $19P$ zu berechnen sind damit nur die folgenden fünf Berechnungen notwendig:

$$2P, 4P = 2P + 2P, 8P = 4P + 4P, 16P = 8P + 8P, 19P = 16P + 2P + P.$$

Diese Methode ermöglicht es, kP für sehr große Wert von k (e.g. mehrere hundert Stellen) sehr schnell zu berechnen. Die einzige Schwierigkeit liegt darin, dass die Größe der Koordinaten der Punkte sehr schnell anwächst, wenn man über den rationalen Zahlen arbeitet. Betrachtet man aber elliptische Kurven über endlichen Körpern (siehe Kapitel [11.3]), dann ist dies kein Problem, da wir stets mod p reduzieren können und damit die Koordinatenwerte der Punkte relativ klein belassen können. Das Assoziativgesetz ermöglicht außerdem eine beliebige Reihenfolge der Berechnungen, um das Endresultat zu erhalten.

Der folgende Algorithmus gibt diese Methode im Detail an.

Algorithmus Ganze Zahl mal einem Punkt

Sei k eine positive ganze Zahl und sei P ein Punkt auf einer elliptischen Kurve. Das folgende Verfahren berechnet kP:

① Input: $k \in \mathbb{N}, P \in E$, Hilfsvariable: eine ganze Zahl a und zwei Punkte B, C.

② Initialisierung: Setze

$$\begin{aligned} a &\longleftarrow k \\ B &\longleftarrow \infty \\ C &\longleftarrow P \end{aligned}$$

③ Wenn a gerade ist, setze:

$$\begin{aligned} a &\longleftarrow a/2 \\ B &\longleftarrow B \\ C &\longleftarrow 2C \end{aligned}$$

④ Wenn a ungerade ist, setze:

$$\begin{aligned} a &\longleftarrow a-1 \\ B &\longleftarrow B+C \\ C &\longleftarrow C \end{aligned}$$

⑤ Wenn $a \neq 0$ gehe zu Schritt ③, sonst: STOP, Output $B = kP$.

Umgekehrt, arbeitet man über einem endlichen Körper $\mathbb{Z}_p$ und p ist hinreichend groß (z.B. 500 Bit) und P und kP sind gegeben, dann ist es sehr schwer, den Wert k zu finden. Dies nennt man das **diskrete Logarithmus Problem für elliptische Kurven**, es bildet die Grundlage kryptographischer Anwendungen.

11.3 Elliptische Kurven über $\mathbb{Z}_p$

Sei $p > 3$ eine Primzahl. Wie wir in Kapitel [15.5] gesehen haben, ist die Menge

$$\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$$

mit den Operationen Addition modulo n und Multiplikation modulo n die algebraische Struktur eines Körpers hat. Auf $\mathbb{Z}_p$ können elliptische Kurven in analoger Weise definiert werden, wie über $\mathbb{R}$, vorausgesetzt die Operationen werden entsprechend angepaßt.

Definition:

Sei $p > 3$ eine Primzahl. Die elliptische Kurve

$$y^2 = x^3 + ax + b$$

über $\mathbb{Z}_p$ ist die Menge der Lösungen $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ der Kongruenz

$$y^2 \equiv x^3 + ax + b \bmod p$$

mit $a, b \in \mathbb{Z}_p$, $a, b = const.$ und

$$4a^3 + 27b^2 \neq 0 \bmod p$$

zusammen mit dem Punkt $P = \infty$, genannt Unendlich.

Additionsoperation über E

Die Additionsoperation auf der elliptischen Kurve E über dem Zahlenkörper $\mathbb{Z}_p$ ist folgendermaßen definiert, sämtliche Operationen sind modulo p.

Seien die Punkte $P = (x_1, y_1)$ und $Q = (x_2, y_2)$, $P, Q \in E$ gegeben. Falls $x_1 = x_2$ und $y_1 = -y_2$, dann definieren wir:

$$P + Q \stackrel{\text{def}}{=} \infty.$$

Andernfalls:

$$P + Q = R = (x_3, y_3)$$

mit

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

wobei die 'Steigung' λ gegeben ist durch:

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{falls } P \neq Q \\ (3x_1^2 + a)(2y)^{-1} & \text{falls } P = Q \end{cases}$$

mit

$$(x_2 - x_1) \cdot (x_2 - x_1)^{-1} \equiv 1 \bmod p$$
$$2y_1 \cdot (2y_1)^{-1} \equiv 1 \bmod p$$

und

$$P + \infty = \infty + P = P \quad \forall\, P \in E$$

Anmerkungen:

① Die formale Übertragung der elliptischen Kurven über $\mathbb{R}$ zu elliptischen Kurven über $\mathbb{Z}_p$ hat zur Folge, daß die anschauliche geometrische Interpretation, die diese Konstruktion der Addition im Reellen hat, verloren geht.

② Die beiden Fälle $p = 2$ und $p = 3$ müssen gesondert betrachtet werden.

Beispiel:

In diesem ausführlichen Beispiel betrachten wir die elliptische Kurve

$$E \;:\; y^2 \equiv (x^3 + 2x + 3) \bmod 5$$

über der Zahlenmenge

$$\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}.$$

Die Punkte auf E sind die Zahlennpaare $(x, y) \in \mathbb{Z}_5 \times \mathbb{Z}_5$, die die obige Gleichung erfüllen zusammen mit dem Punkt im Unendlichen.

Da $a = 2, b = 3$ gilt:

$$\begin{aligned} 4a^3 + 27b^2 &= 32 + 243 \\ &= 2 + 3 \\ &\equiv 0 \bmod 5 \end{aligned}$$

ist die elliptische Kurve singulär.

Die Bestimmung der Punkte auf der elliptischen Kurve geschieht folgendermaßen. $x \in \mathbb{Z}_5$ kann die Werte

$$x = 0, 1, 2, 3, 4$$

annehmen. Diese Werte setzt man nun sukzessive in die Gleichung

$$y^2 \equiv (x^3 + 2x + 3) \bmod 5$$

ein und versucht, quadratische Reste zu finden.

1. $x = 0$ damit:

$$y^2 \equiv 3 \bmod 5$$

Diese Gleichung hat keine Lösung, i.e. es gibt kein $y \in \mathbb{Z}_5$ mit $y^2 \equiv 3 \bmod 5$.

2. $x = 1$. Einsetzen liefert

$$\begin{aligned} y^2 &\equiv 1 + 2 + 3 \bmod 5 \\ &\equiv 6 \bmod 5 \\ &\equiv 1 \bmod 5 \end{aligned}$$

mit Lösung

$$y = 1, 4$$

denn

$$\begin{aligned} 1^2 &\equiv 1 \bmod 5 \\ 4^2 &\equiv 16 \bmod 5 \equiv 1 \bmod 5 \end{aligned}$$

3. $x = 2$. Einsetzen liefert

$$\begin{aligned} y^2 &\equiv (8 + 4 + 3) \bmod 5 \\ &\equiv 15 \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Lösung: $y = 0$

4. $x = 3$. Einsetzen liefert

$$\begin{aligned} y^2 &\equiv (27 + 6 + 3) \bmod 5 \\ &\equiv 36 \bmod 5 \\ &\equiv 1 \bmod 5 \end{aligned}$$

Lösungen: $y = 1, 4$

5. $x = 4$. Einsetzen liefert

$$\begin{aligned} y^2 &\equiv (64 + 8 + 3) \bmod 5 \\ &\equiv 75 \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Lösung: $y = 0$

Damit sind die Punkte auf der elliptischen Kurve

$$E \; : y^2 \equiv (x^3 + 2x + 3) \bmod 5$$

die Menge

$$\mathbb{L} = \big\{(1,1), (1,4), (2,0), (3,1), (3,4), (4,0), (\infty, \infty)\big\} \subset \mathbb{Z}_5 \times \mathbb{Z}_5$$

Die Additionsoperation auf dieser Menge ist:

$$\begin{aligned} + \; &: \mathbb{L} \times \mathbb{L} \longrightarrow \mathbb{L} \\ &(P, Q) \quad \longmapsto \; R = (x_3, y_3) = P + Q = (x_1, y_1) + (x_2, y_2) \end{aligned}$$

mit

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$
$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{falls } P \neq Q \\ (3x_1^2 + a)(2y_1)^{-1} & \text{falls } P = Q \end{cases}$$

Um etwas mehr Vertrautheit mit dem Thema zu erhalten, ist es sinnvoll, die komplette Additionstafel auf dieser Menge $\mathbb{L}$ zu berechnen. Diese Gruppentafel ist in der Tabelle [11.2] aufgelistet.

+	(1,1)	(1,4)	(2,0)	(3,1)	(3,4)	(4,0)	∞
(1,1)	(3,4)	∞	(3,1)	(1,4)	(2,0)	(4,0)	(1,1)
(1,4)	∞	(3,1)	(3,4)	(2,0)	(1,1)	(4,0)	(1,4)
(2,0)	(3,1)	(3,4)	∞	(4,0)	(1,4)	(4,0)	(2,0)
(3,1)	(1,4)	(2,0)	(4,0)	(3,4)	∞	(4,0)	(3,1)
(3,4)	(2,0)	(1,1)	(1,4)	∞	(3,4)	(4,0)	(3,4)
(4,0)	(4,0)	(4,0)	(4,0)	(4,0)	(4,0)	∞	(4,0)
∞	(1,1)	(1,4)	(2,0)	(3,1)	(3,4)	(4,0)	∞

Tabelle 11.2: Gruppentafel.

1. $P = Q = (1, 1)$; damit ist der Fall $P = Q$ zu betrachten mit

   $$2P = R$$

   $R = (x_3, y_3)$ mit den Koordinaten

   $$x_3 = \lambda^2 - 2x_1$$
   $$y_3 = \lambda(x_1 - x_3) - y_1$$

   und der Steigung

   $$\begin{aligned} \lambda &= (3x_1 + a)(2y_1)^{-1} \\ &\equiv (3 \cdot 1 + 2)(2 \cdot 1)^{-1} \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

   Damit ist

   $$x_3 = -2x_1 = -2 \equiv 3 \bmod 5$$
   $$y_3 = -y_1 = -1 \equiv 4 \bmod 5$$

   Daher

   $$2P = (1, 1) + (1, 1) = R = (3, 4)$$

2. $P = (1, 1), Q = (1, 4)$; damit ist der Fall $P = -Q$ zu betrachten, denn $x_1 = x_2$ und $y_1 = -y_2$, denn

   $$1 \equiv -4 \bmod 5$$

Dieser Fall ist definiert durch

$$P * Q = (1,1) + (1,4) = \infty$$

3. $P = (1,1), Q = (2,0)$; damit ist der Fall $P \neq Q$ zu betrachten. Setze

$$x_1 = 1; y_1 = 1$$
$$x_2 = 2; y_2 = 0$$

Steigung:

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (0-1)(2-1)^{-1} \\ &= (-1)(1)^{-1} \\ &\equiv 4 \bmod 5\end{aligned}$$

Damit:

$$\begin{aligned}x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv (16 - 1 - 2) \bmod 5 \\ &\equiv 13 \bmod 5 \\ &\equiv 3 \bmod 5\end{aligned}$$

und

$$\begin{aligned}y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv 4 \cdot (1-3) - 1 \bmod 5 \\ &\equiv -9 \bmod 5 \\ &\equiv 1 \bmod 5\end{aligned}$$

Daher

$$(1,1) + (2,0) = (3,1) = R$$

4. $P = (1,1), Q = (3,1)$; damit ist der Fall $P \neq Q$ zu betrachten. Setze

$$x_1 = 1; y_1 = 1$$
$$x_2 = 3; y_2 = 1$$

Steigung:

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (1-1)(3-1)^{-1} \\ &\equiv 0 \bmod 5\end{aligned}$$

Damit:

$$\begin{aligned}x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv (-3 - 1) \bmod 5 \\ &\equiv -4 \bmod 5 \\ &\equiv 1 \bmod 5\end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv -1 \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

Daher

$$(1,1) + (3,1) = (1,4) = R$$

5. $P = (1,1), Q = (3,4)$; damit ist der Fall $P \neq Q$ zu betrachten. Setze

$$\begin{aligned} x_1 = 1; y_1 = 1 \\ x_2 = 3; y_2 = 4 \end{aligned}$$

Steigung:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (4-1)(3-1)^{-1} \\ &\equiv 3 \cdot (2)^{-1} \bmod 5 \\ &\equiv 3 \cdot 3 \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv (16 - 1 - 3) \bmod 5 \\ &\equiv 12 \bmod 5 \\ &\equiv 2 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv 4 \cdot (1-2) - 1 \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Daher

$$(1,1) + (3,4) = (2,0) = R$$

6. $P = (1,1), Q = (4,0)$; damit ist der Fall $P \neq Q$ zu betrachten. Setze

$$\begin{aligned} x_1 = 1; y_1 = 1 \\ x_2 = 4; y_2 = 0 \end{aligned}$$

Steigung:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (0-1)(4-1)^{-1} \\ &\equiv (-1) \cdot (3)^{-1} \bmod 5 \\ &\equiv 4 \cdot 2 \bmod 5 \\ &\equiv 3 \bmod 5 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv (9 - 1 - 4) \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv 3 \cdot (1 - 4) - 1 \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Daher

$$(1,1) + (4,0) = (4,0) = R$$

7. $P = Q = (1,4)$; damit ist der Fall $P = Q$ zu betrachten mit

$$2P = R$$

$R = (x_3, y_3)$ mit den Koordinaten

$$\begin{aligned} x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

und der Steigung

$$\begin{aligned} \lambda &= (3x_1 + a)(2y_1)^{-1} \\ &\equiv (3 \cdot 1 + 2)(2 \cdot 4)^{-1} \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Damit ist

$$\begin{aligned} x_3 &= -2x_1 = -2 \equiv 3 \bmod 5 \\ y_3 &= -y_1 = -4 \equiv 1 \bmod 5 \end{aligned}$$

Daher

$$2P = (1,4) + (1,4) = R = (3,1)$$

8. $P = (1,4), Q = (2,0)$; damit ist der Fall $P \neq Q$ zu betrachten, mit

$$\begin{aligned} x_1 &= 1; y_1 = 4 \\ x_2 &= 2; y_2 = 0 \end{aligned}$$

Die Steigung ist für diesen Fall:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (-4)(2 - 1)^{-1} \\ &\equiv 1 \bmod 5 \end{aligned}$$

da

$$(1)^{-1} \cdot 1 \equiv 1 \bmod 5$$

Daher ist

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= (1 - 1 - 2) \bmod 5 \\ &\equiv -2 \bmod 5 \\ &\equiv 3 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= 1(1 - 3) - 4 \bmod 5 \\ &\equiv -6 \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

Daher

$$(1, 4) + (2, 0) = R = (3, 4)$$

9. $P = (1, 4), Q = (3, 1)$; damit ist der Fall $P \neq Q$ zu betrachten mit

$$\begin{aligned} x_1 = 1; y_1 = 4 \\ x_2 = 3; y_2 = 1 \end{aligned}$$

Steigung:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (1 - 4)(3 - 1)^{-1} \\ &= (-3)(2)^{-1} \\ &= 2 \cdot 3 \\ &\equiv 1 \bmod 5 \end{aligned}$$

wobei wir nutzen:

$$2^{-1} \equiv 3 \bmod 5$$

denn

$$2 \cdot 3 = 6 \equiv 1 \bmod 5$$

Damit ist also

$$\lambda \equiv 1 \bmod 5$$

Mit diesem Wert folgt:

$$\begin{aligned} x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv (1 - 1 - 3) \bmod 5 \\ &\equiv -3 \bmod 5 \\ &\equiv 2 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv 1 \cdot (1-2) - 4 \bmod 5 \\ &\equiv -5 \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Damit folgt

$$\begin{aligned} R &= P + Q \\ &= (x_1, y_1) + (x_2, y_2) \\ &= (1,4) + (3,1) \\ &= (2,0) \end{aligned}$$

10. $P = (1,4), Q = (3,4)$; damit ist der Fall $P \neq Q$ zu betrachten, mit

$$\begin{aligned} x_1 = 1; y_1 = 4 \\ x_2 = 3; y_2 = 4 \end{aligned}$$

Steigung:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &\equiv 0 \bmod 5 \end{aligned}$$

Mit diesem Wert folgt:

$$\begin{aligned} x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv -1 - 3 \bmod 5 \\ &\equiv -4 \bmod 5 \\ &\equiv 1 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv -4 \bmod 5 \\ &\equiv 1 \bmod 5 \end{aligned}$$

Damit folgt

$$(1,4) + (3,4) = R = (1,1)$$

11. $P = (1,4), Q = (4,0)$; damit ist der Fall $P \neq Q$ zu betrachten. Setze

$$\begin{aligned} x_1 = 1; y_1 = 4 \\ x_2 = 4; y_2 = 0 \end{aligned}$$

Steigung:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (0-4)(4-1)^{-1} \\ &\equiv (1) \cdot (3)^{-1} \bmod 5 \\ &\equiv 1 \cdot 2 \bmod 5 \\ &\equiv 2 \bmod 5 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv (4 - 1 - 4) \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv 2 \cdot (1 - 4) - 4 \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Daher

$$(1,1) + (4,0) = (4,0)$$

12. $P = Q = (2,0)$; damit ist der Fall

13. $P = (2,0), Q = (3,1)$; damit ist der Fall $P \neq Q$ zu betrachten, mit

$$x_1 = 2; y_1 = 0 x_2 = 3; \qquad y_2 = 1$$

und

$$R = (x_3, y_3) = (x_1, y_1) + (x_2, y_2)$$

mit

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= (y_2 - y_1) \cdot (x_2 - x_1)^{-1} \end{aligned}$$

Die Steigung ist für diesen Fall:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (0 - 1)(3 - 2)^{-1} \\ &\equiv (-1) \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

Daher ist

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= (16 - 3 - 2) \bmod 5 \\ &\equiv 9 \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= 4(3-4) - 1 \bmod 5 \\ &\equiv 0 \bmod 5 \end{aligned}$$

Daher

$$(2,0) + (3,1) = (4,0) = R$$

14. $P = (2,0), Q = (3,4)$; damit ist der Fall $P \neq Q$ zu betrachten, mit

$$\begin{aligned} x_1 &= 2; y_1 = 0 \\ x_2 &= 3; y_2 = 4 \end{aligned}$$

Steigung:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &\equiv 4 \cdot 1^{-1} \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

Mit diesem Wert folgt:

$$\begin{aligned} x_3 &= (\lambda^2 - x_1 - x_2) \bmod 5 \\ &\equiv 16 - 2 - 3 \bmod 5 \\ &\equiv 11 \bmod 5 \\ &\equiv 1 \bmod 5 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 5 \\ &\equiv 4(2-1) \bmod 5 \\ &\equiv 4 \bmod 5 \end{aligned}$$

Damit folgt

$$(2,0) + (3,4) = (1,4)$$

Beispiel:

In einem zweiten Beispiel untersuchen wir die elliptische Kurve

$$E : y^2 \equiv x^3 + x + 6 \bmod 11$$

über $\mathbb{Z}_{11}$.

Wir bestimmen zunächst die Punkte auf E. Dies kann folgendermaßen durchgeführt werden: Für jedes $x \in \mathbb{Z}_{11}$ berechnet man $x^3 + x + 6 \bmod 11$ und versucht dann, die Gleichung

$$y^2 \equiv x^3 + x + 6 \bmod 11$$

nach y zu lösen. Dies kann man beispielsweise durch anwenden des EULER Kriteriums durchführen und testen, ob

$$z = x^3 + x + 6 \bmod 11$$

ein quadratischer Rest ist. Wie wir in Abschnitt [17.3] abgeleitet haben, gibt es eine explizite Formel um Quadratwurzeln quadratischer Reste zu bestimmen, wenn die Primzahl p die Bedingung

$$p \equiv 3 \bmod 4$$

erfüllt, was in unserem Beispiel der Fall ist. Wendet man diese Formel an, erhält man die Quadratwurzeln der quadratischen Reste z zu:

$$\pm z^{\frac{11+1}{4}} \bmod 11 \equiv \pm z^3 \bmod 11$$

Also: Die Vorgehensweise besteht aus folgenden Schritten:

1. Für $x = 0, 1, \ldots, 10$ führe folgende Steps aus:
2. Bestimme
$$z = x^3 + x + 6 \bmod 11$$
3. Teste mit dem EULER Kriterium, ob z quadratischer Rest ist, i.e. prüfe, ob
$$z^{\frac{p-1}{2}} \bmod p \equiv 1 \bmod p$$
ist.
4. Wenn ja, berechne
$$z^{\frac{p+1}{4}} \bmod p$$
und $p - z$.

Die folgende Tabelle liefert die Resultate dieser Rechnungen:

x	$x^3 + x + 6 \bmod 11$	Quadr. Rest?	y
0	6	nein	
1	8	nein	
2	5	ja	4,7
3	3	ja	5,6
4	8	nein	
5	4	ja	2,9
6	8	nein	
7	4	ja	2,9
8	9	ja	3,8
9	7	nein	
10	4	ja	2,9

Damit hat die elliptische Kurve

$$E : y^2 \equiv x^3 + x + 6 \bmod 11$$

die Punkte

$$\mathbb{L} = \{(2,4),(2,7),(3,5),(3,6),(5,2),(5,9),\\ (7,2),(7,9),(8,3),(8,8),(10,2),(10,9),\infty\}$$

Damit hat die Kurve E 13 Punkte. Da jede Gruppe, deren Ordnung eine Primzahl ist, zyklisch ist, folgt, daß E isomorph zu $\mathbb{Z}_{13}$ ist. Jeder Punkt der Kurve – bis auf ∞ – ist Erzeuger von E, das bedeutet, die Vielfachen des Erzeugers generieren die komplette Gruppe. Als Erzeuger nehmen wir den Punkt

$$\alpha = (2,7)$$

Wir zeigen:

$\alpha = (2,7)$	$7\alpha = (7,2)$
$2\alpha = (5,2)$	$8\alpha = (3,5)$
$3\alpha = (8,3)$	$9\alpha = (10,9)$
$4\alpha = (10,2)$	$10\alpha = (8,8)$
$5\alpha = (3,6)$	$11\alpha = (5,9)$
$6\alpha = (7,9)$	$12\alpha = (2,4)$

Nebenrechnung:

① Berechnung von $2\alpha = (2,7) + (2,7)$:
Wir berechnen zunächst:

$$\begin{aligned}\lambda &= (3x^2 + a)(2y)^{-1} \bmod 11\\ &= (3 \cdot 2^2 + 1)(2 \cdot 7)^{-1} \bmod 11\\ &\equiv (2 \cdot 3^{-1}) \bmod 11\\ &\equiv 2 \cdot 4 \bmod 11\\ &\equiv 8 \bmod 11\end{aligned}$$

Damit erhalten wir:

$$\begin{aligned}x_3 &= \lambda^2 - 2x \bmod 11\\ &= 64 - 4 \bmod 11\\ &= 5 \bmod 11\end{aligned}$$

und

$$\begin{aligned}y_3 &= \lambda \cdot (x - x_3) - y\\ &= 8 \cdot (2 - 5) - 7 \bmod 11\\ &= 2 \bmod 11\end{aligned}$$

Damit:

$$2\alpha = (5,2)$$

② Berechnung von $3\alpha = (2,7) + (5,2)$:
Zunächst berechnen wir wieder die Steigung λ. In diesem Fall sind die Punkte unterschiedlich, daher

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11\\ &= (2-7)(5-2)^{-1} \bmod 11\\ &\equiv 6 \cdot 3^{-1} \bmod 11\\ &\equiv 2 \bmod 11\end{aligned}$$

Damit:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2 \bmod 11\\ &= 4 - 2 - 5 \bmod 11\\ &\equiv 8 \bmod 11\end{aligned}$$

und

$$\begin{aligned}y_3 &= \lambda(x_1 - x_3) - y_1 \bmod 11\\ &= 2(2-8) - 7 \bmod 11\\ &\equiv 3 \bmod 11\end{aligned}$$

Damit:

$$3\alpha = (8,3)$$

③ Berechnung von $4\alpha = (2,7) + (8,3)$:
Wir setzen:

$$\begin{aligned}P &= (x_1, y_1) = (2,7)\\ Q &= (x_2, y_2) = (8,3)\end{aligned}$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2\\ y_3 &= \lambda \cdot (x_1 - x_3) - y_1\end{aligned}$$

Die Steigung ist

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11\\ &= (3-7)(8-2)^{-1} \bmod 11\\ &\equiv 7 \cdot 6^{-1} \bmod 11\\ &\equiv 7 \cdot 2 \bmod 11\\ &\equiv 3 \bmod 11\end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 9 - 2 - 8 \bmod 11 \\ &= -1 \bmod 11 \\ &\equiv 10 \bmod 11 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 3 \cdot (2 - 10) - 7 \bmod 11 \\ &\equiv 3 \cdot 3 - 7 \bmod 11 \\ &\equiv 2 \bmod 11 \end{aligned}$$

Damit:

$$4\alpha = (10, 2)$$

④ Berechnung von $5\alpha = (2, 7) + (10, 2)$:
Wir setzen:

$$\begin{aligned} P &= (x_1, y_1) = (2, 7) \\ Q &= (x_2, y_2) = (10, 2) \end{aligned}$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \end{aligned}$$

Die Steigung ist

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11 \\ &= (2 - 7)(10 - 2)^{-1} \bmod 11 \\ &\equiv 6 \cdot 8^{-1} \bmod 11 \\ &\equiv 6 \cdot 7 \bmod 11 \\ &\equiv 9 \bmod 11 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 81 - 2 - 10 \bmod 11 \\ &= 69 \bmod 11 \\ &\equiv 3 \bmod 11 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 9 \cdot (2-3) - 7 \bmod 11 \\ &\equiv 9 \cdot 10 - 7 \bmod 11 \\ &\equiv 6 \bmod 11 \end{aligned}$$

Damit:

$$5\alpha = (3,6)$$

⑤ Berechnung von $6\alpha = (2,7) + (3,6)$:
Wir setzen:

$$\begin{aligned} P &= (x_1, y_1) = (2,7) \\ Q &= (x_2, y_2) = (3,6) \end{aligned}$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \end{aligned}$$

Die Steigung ist

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11 \\ &= (6-7)(3-2)^{-1} \bmod 11 \\ &\equiv (-1) \cdot 1^{-1} \bmod 11 \\ &\equiv 10 \cdot 1 \bmod 11 \\ &\equiv 10 \bmod 11 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 100 - 2 - 3 \bmod 11 \\ &= 95 \bmod 11 \\ &\equiv 7 \bmod 11 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 10 \cdot (2-7) - 7 \bmod 11 \\ &\equiv 10 \cdot 6 - 7 \bmod 11 \\ &\equiv 9 \bmod 11 \end{aligned}$$

Damit:

$$6\alpha = (7,9)$$

⑥ Berechnung von $7\alpha = (2,7) + (7,9)$:
Wir setzen:

$$P = (x_1, y_1) = (2,7)$$
$$Q = (x_2, y_2) = (7,9)$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

Die Steigung ist

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11 \\ &= (9-7)(7-2)^{-1} \bmod 11 \\ &\equiv 2 \cdot 5^{-1} \bmod 11 \\ &\equiv 2 \cdot 9 \bmod 11 \\ &\equiv 7 \bmod 11\end{aligned}$$

Damit:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2 \\ &= 49 - 2 - 7 \bmod 11 \\ &\equiv 7 \bmod 11\end{aligned}$$

und

$$\begin{aligned}y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 7(2-7) - 7 \bmod 11 \\ &\equiv 7 \cdot 6 - 7 \bmod 11 \\ &\equiv 2 \bmod 11\end{aligned}$$

Damit:

$$7\alpha = (7,2)$$

⑦ Berechnung von $8\alpha = (2,7) + (7,2)$:
Wir setzen:

$$P = (x_1, y_1) = (2,7)$$
$$Q = (x_2, y_2) = (7,2)$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

Die Steigung ist

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11 \\ &= (2-7)(7-2)^{-1} \bmod 11 \\ &\equiv 6 \cdot 5^{-1} \bmod 11 \\ &\equiv 6 \cdot 9 \bmod 11 \\ &\equiv 10 \bmod 11\end{aligned}$$

Damit:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2 \\ &= 100 - 2 - 7 \bmod 11 \\ &\equiv 3 \bmod 11\end{aligned}$$

und

$$\begin{aligned}y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 10(2-3) - 7 \bmod 11 \\ &\equiv 5 \bmod 11\end{aligned}$$

Damit:

$$8\alpha = (3,5)$$

⑧ Berechnung von $9\alpha = (2,7) + (3,5)$:
Wir setzen:

$$P = (x_1, y_1) = (2,7)$$
$$Q = (x_2, y_2) = (3,5)$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

Die Steigung ist

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11 \\ &= (5-7)(3-2)^{-1} \bmod 11 \\ &\equiv 9 \cdot 1^{-1} \bmod 11 \\ &\equiv 9 \bmod 11\end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 81 - 2 - 3 \bmod 11 \\ &\equiv 10 \bmod 11 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 9(2 - 10) - 7 \bmod 11 \\ &\equiv 9 \bmod 11 \end{aligned}$$

Damit:

$$9\alpha = (10, 9)$$

⑨ Berechnung von $10\alpha = (2, 7) + (10, 9)$:
Wir setzen:

$$\begin{aligned} P &= (x_1, y_1) = (2, 7) \\ Q &= (x_2, y_2) = (3, 5) \end{aligned}$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \end{aligned}$$

Die Steigung ist

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11 \\ &= (9 - 7)(10 - 2)^{-1} \bmod 11 \\ &\equiv 2 \cdot 8^{-1} \bmod 11 \\ &\equiv 3 \bmod 11 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 9 - 2 - 10 \bmod 11 \\ &\equiv 8 \bmod 11 \end{aligned}$$

und

$$\begin{aligned} y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 3(2 - 8) - 7 \bmod 11 \\ &\equiv 8 \bmod 11 \end{aligned}$$

Damit:

$$10\alpha = (8, 8)$$

⑩ Berechnung von $11\alpha = (2,7) + (8,8)$:
Wir setzen:

$$P = (x_1, y_1) = (2,7)$$
$$Q = (x_2, y_2) = (8,8)$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

Die Steigung ist

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11 \\ &= (8-7)(8-2)^{-1} \bmod 11 \\ &\equiv 1 \cdot 6^{-1} \bmod 11 \\ &\equiv 2 \bmod 11\end{aligned}$$

Damit:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2 \\ &= 4 - 2 - 8 \bmod 11 \\ &\equiv 5 \bmod 11\end{aligned}$$

und

$$\begin{aligned}y_3 &= \lambda \cdot (x_1 - x_3) - y_1 \\ &= 2(2-5) - 7 \bmod 11 \\ &\equiv 9 \bmod 11\end{aligned}$$

Damit:

$$11\alpha = (5,9)$$

❶ Berechnung von $12\alpha = (2,7) + (5,9)$:
Wir setzen:

$$P = (x_1, y_1) = (2,7)$$
$$Q = (x_2, y_2) = (5,9)$$

Da $P \neq Q$ ist:

$$R = (x_3, y_3) \stackrel{!}{=} P + Q$$

mit:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

Die Steigung ist

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \bmod 11\\ &= (9-7)(5-2)^{-1} \bmod 11\\ &\equiv 2 \cdot 3^{-1} \bmod 11\\ &\equiv 8 \bmod 11\end{aligned}$$

Damit:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2\\ &= 64 - 2 - 5 \bmod 11\\ &\equiv 2 \bmod 11\end{aligned}$$

und

$$\begin{aligned}y_3 &= \lambda \cdot (x_1 - x_3) - y_1\\ &= 8(2-2) - 7 \bmod 11\\ &\equiv 4 \bmod 11\end{aligned}$$

Damit:

$$12\alpha = (2, 4)$$

11.4 Faktorisierung mit elliptischen Kurven

Angenommen, das Produkt $n = p \cdot q$ ist eine Zahl, die faktorisiert werden soll. Die Faktorisierung dieser Zahl mit Hilfe elliptischer Kurven beginnt damit, dass

① eine zufällige elliptische Kurve

$$E \; : \; y^2 \equiv x^3 + ax + b \bmod n$$

② und ein beliebiger Punkt P auf E

gewählt wird. In der Praxis wählt man beispielsweise 14 zufällige Kurven, wenn eine (dezimal) 50stellige Zahl zu faktorisieren ist. Bei größeren Zahlen wählt man eine entsprechend größere Anzahl von Kurven und läßt den Algorithmus zur Faktorisierung parallel laufen.

Die Festlegung der Kurve geschieht folgendermaßen. Zunächst wird ein Punkt P gewählt sowie der Parameter a festgelegt. Anschließend wird der zweite Parameter b der Kurve so fixiert, daß P auf der Kurve

$$E \; : \; y^2 \equiv x^3 + ax + b \bmod n$$

liegt. Dieses Verfahren ist wesentlich effizienter als zuerst die Kurve durch die Wahl der Koeffizienten a, b festzulegen und dann einen Punkt P zu finden.

Beispiel:

Sei $n = 2773$; wir wählen $P = (1, 3) = (x, y)$ und $a = 4$. Da

$$y^2 \equiv x^3 + ax + b \bmod n$$

setzt man einfach die Werte von P und a ein und erhält:

$$3^2 \equiv 1 + 4 \cdot 1 + b \bmod 2773$$

oder

$$9 \equiv 5 + b \bmod 2773$$

Damit ist $b = 4$ und die gesuchte elliptische Kurve ist

$$\boxed{y^2 \equiv x^3 + 4x + 4 \bmod 2773}$$

Von dem gewählten Punkt $p = (1, 3)$ ausgehend, berechnet man nun sukzessive die Vielfachen von P.

$$2P = P + P = (1, 3) + (1, 3)$$

Um die Additionsoperation zu berechnen ist hier der Fall $P = Q$ anzuwenden mit $x_1 = 1, y_1 = 3$ und $a = 4$.

$$\begin{aligned} x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= (3x_1^2 + a)(2y_1)^{-1} \end{aligned}$$

Die Steigung in $P = (1, 3)$ berechnet sich aus

$$\lambda \equiv (3 + 4)(2 \cdot 3)^{-1} \bmod 2773$$

i.e. wir müssen das multiplikative Inverse von 6 modulo 2773 berechnen.
Gesucht ist eine Zahl $x \in \mathbb{Z}_{2773}$ mit

$$6 \cdot x \equiv 1 \bmod 2773$$

Die Zahl x findet man durch anwenden des erweiterten EUKLIDischen Algorithmus.

$$\begin{aligned} x_1 &= 1 & y_1 &= 0 \\ x_2 &= 0 & y_2 &= 1 \\ x_3 &= 2773 & y_3 &= 6 \end{aligned}$$

Damit ist

$$Q = \lfloor \frac{x_3}{y_3} \rfloor = 462$$

und

$$\begin{aligned} t_1 &= x_1 - Qy_1 = 1 \\ t_2 &= x_2 - Qy_2 = -462 \\ t_3 &= x_3 - Qy_3 = 1 \end{aligned}$$

Substitution liefert

$$\begin{aligned} x_1 &\longleftarrow 0 & y_1 &\longleftarrow 1 \\ x_2 &\longleftarrow 1 & y_2 &\longleftarrow -462 \\ x_3 &\longleftarrow 6 & y_3 &\longleftarrow 1 \end{aligned}$$

Da $y_3 = 1$ terminiert der Algorithmus, daher

$$x = -462 \equiv 2311 \bmod 2773$$

Check:

$$6 \cdot 2773 = 13866 = 5 \cdot 2773 + 1$$

Damit ist die Steigung

$$\lambda \equiv 7 \cdot 2311 \bmod 2773 = 2312 \bmod 2773$$

Damit erhalten wir:

$$\begin{aligned} x_3 &\equiv \lambda^2 - 2x_1 \bmod 2773 \\ &\equiv 2312^2 - 2 \bmod 2773 \\ &\equiv 1771 \bmod 2773 \end{aligned}$$

und

$$\begin{aligned} y_3 &\equiv \lambda(x_1 - x_3) - y_1 \bmod 2773 \\ &\equiv 2312 \cdot (1 - 1771) - 3 \bmod 2773 \\ &\equiv 2312 \cdot 1003 - 3 \bmod 2773 \\ &\equiv 705 \bmod 2773 \end{aligned}$$

Damit:

$$2P = (1771, 705)$$

Bei der nächsten Iteration ist zu berechnen:

$$3P = 2P + P = (1771, 705) + (1, 3)$$

Hier ist der Fall $P \neq Q$ anzuwenden, mit

$$\begin{aligned} x_1 &= 1771 \quad y_1 = 705 \\ x_2 &= 1 \quad y_2 = 3 \end{aligned}$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \end{aligned}$$

Die Steigung ist:

$$\begin{aligned} \lambda &\equiv (y_2 - y_1)^{-1} \bmod 2773 \\ &\equiv (3 - 705) \cdot (1 - 1771)^{-1} \bmod 2773 \\ &\equiv (-702) \cdot (-1770)^{-1} \bmod 2773 \\ &\equiv 702 \cdot (1770)^{-1} \bmod 2773 \end{aligned}$$

Gesucht ist eine Zahl $x \in \mathbb{Z}_{2773}$ mit

$$1770 \cdot x \equiv 1 \bmod 2773$$

Wir wenden den erweiterten EUKLIDischen Algorithmus an, um x zu bestimmen.

$$\begin{aligned} x_1 &= 1 & y_1 &= 0 \\ x_2 &= 0 & y_2 &= 1 \\ x_3 &= 2773 & y_3 &= 1770 \end{aligned}$$

Damit ist

$$Q = \lfloor \frac{x_3}{y_3} \rfloor = 1$$

und

$$\begin{aligned} t_1 &= x_1 - Qy_1 = 1 \\ t_2 &= x_2 - Qy_2 = -1 \\ t_3 &= x_3 - Qy_3 = 1003 \end{aligned}$$

Substitution liefert

$$\begin{aligned} x_1 &\longleftarrow 0 & y_1 &\longleftarrow 1 \\ x_2 &\longleftarrow 1 & y_2 &\longleftarrow -1 \\ x_3 &\longleftarrow 1770 & y_3 &\longleftarrow 1003 \end{aligned}$$

Da $y_3 \neq 0, 1$:

$$Q = \lfloor \frac{x_3}{y_3} \rfloor = 1$$

und

$$\begin{aligned} t_1 &= x_1 - Qy_1 = -1 \\ t_2 &= x_2 - Qy_2 = 2 \\ t_3 &= x_3 - Qy_3 = 767 \end{aligned}$$

Substitution liefert

$$\begin{aligned} x_1 &\longleftarrow 1 & y_1 &\longleftarrow -1 \\ x_2 &\longleftarrow -1 & y_2 &\longleftarrow 2 \\ x_3 &\longleftarrow 1003 & y_3 &\longleftarrow 767 \end{aligned}$$

Nach dieser Iteration ist $y_3 \neq 0, 1$:

$$Q = \lfloor \frac{x_3}{y_3} \rfloor = \lfloor \frac{1003}{767} \rfloor = 1$$

und

$$\begin{aligned} t_1 &= x_1 - Qy_1 = 2 \\ t_2 &= x_2 - Qy_2 = -3 \\ t_3 &= x_3 - Qy_3 = 236 \end{aligned}$$

Substitution liefert

$$\begin{aligned} x_1 &\longleftarrow -1 & y_1 &\longleftarrow 2 \\ x_2 &\longleftarrow 2 & y_2 &\longleftarrow -3 \\ x_3 &\longleftarrow 767 & y_3 &\longleftarrow 236 \end{aligned}$$

Nach dieser Iteration ist $y_3 \neq 0, 1$:

$$Q = \lfloor \frac{x_3}{y_3} \rfloor = \lfloor \frac{767}{236} \rfloor = 3$$

und

$$\begin{aligned} t_1 &= x_1 - Qy_1 = -7 \\ t_2 &= x_2 - Qy_2 = 11 \\ t_3 &= x_3 - Qy_3 = 59 \end{aligned}$$

Die neuen Variablen sind

$$\begin{aligned} x_1 &\longleftarrow 2 & y_1 &\longleftarrow -7 \\ x_2 &\longleftarrow -3 & y_2 &\longleftarrow 11 \\ x_3 &\longleftarrow 236 & y_3 &\longleftarrow 59 \end{aligned}$$

Nach dieser Iteration ist $y_3 \neq 0, 1$:

$$Q = \lfloor \frac{x_3}{y_3} \rfloor = \lfloor \frac{236}{59} \rfloor = 4$$

und

$$\begin{aligned} t_1 &= x_1 - Qy_1 = 30 \\ t_2 &= x_2 - Qy_2 = -47 \\ t_3 &= x_3 - Qy_3 = 0 \end{aligned}$$

Die neuen Variablen sind

$$\begin{aligned} x_1 &\longleftarrow -7 & y_1 &\longleftarrow 30 \\ x_2 &\longleftarrow 11 & y_2 &\longleftarrow -47 \\ x_3 &\longleftarrow 59 & y_3 &\longleftarrow 0 \end{aligned}$$

Da $y_3 = 0$, terminiert der Algorithmus. Es gibt kein multiplikatives Inverses 1770 modulo 2773. Ergebnis:

$$\mathrm{ggT}(1770, 2773) = 59.$$

Damit sind wir am Ziel. Unser Ausgangspunkt war die Faktorisierung der Zahl $n = 2773$. Die obige Rechnungen liefern einen Faktor, nämlich $p = 59$, und wir haben die Faktorisierung

$$n = 2773 = 59 \cdot 47 = p \cdot q.$$

11.5 Kryptosysteme auf elliptischen Kurven

Es gibt eine Vielzahl von Kryptosystemen, die auf elliptischen Kurven basieren, insbesondere werden Kryptosysteme, deren Sicherheit auf dem diskreten Logarithmusproblem beruhen wie zum Beispiel das DIFFIE-HELLMAN Key Exchange Verfahren oder das ELGAMAL Kryptosystem, auf elliptische Kurven übertragen. Ein großer Vorteil der Implementierung von Kryptosystemen auf elliptischen Kurven gegenüber der Implementierung auf multiplikativen Gruppen $(\mathbb{Z}_p^*, \cdot)$ ist folgender: Arbeitet man auf multiplikativen Gruppen $(\mathbb{Z}_p^*, \cdot)$ ist es möglich, die Faktorisierung als Angriffsmethode zur Lösung des diskreten Logarithmusproblems zu verwenden. Dies ist der Index Kalkül. Bisher ist keine vergleichbare Methode für elliptische Kurven bekannt. Daher ist es bei der Implementierung auf elliptischen Kurven möglich, wesentlich kleiner Primzahlen oder kleiner endliche Körper zu betrachten, wobei ein vergleichbarer Grad an Sicherheit erreicht wird wie bei der Implementierung mit wesentlich größeren Zahlen modulo p. Dies ermöglicht viel kürzere Schlüssellängen und effektivere Algorithmen.

11.5.1 Codierung von Klartext

Wie wir bereits bei vielen Beispielen gesehen haben, erfordert die Verwendung eines Kryptosystems eine geeignete Codierung des Klartexts in Form numerischer Werte um mathematische Operationen ausführen zu können. Um elliptische Kurven benutzen zu können, benötigt man daher ein systematisches Verfahren, eine Nachricht auf Punkte einer elliptischen Kurve abzubilden. Elliptische Kurven Kryptosysteme benutzen elliptische Kurven Operationen auf diesen Punkt und erzeugen auf diese Weise einen neuen Punkt, der die Rolle des Chiffretextes spielt.

Das Problem der Codierung von Klartext als Punkte auf elliptischen Kurven ist nicht so einfach wie in den bisher betrachteten Fällen (siehe [129, p. 179], [215, pp. 279]). Insbesondere ist kein deterministischer Polynomialzeit Algorithmus[5] bekannt, um Punkte auf eine beliebige elliptische Kurve $E \bmod p$ zu bestimmen. Es gibt jedoch schnelle probabalistische Algorithmen um Punkte auf einer beliebigen elliptischen Kurve zu finden. Diese Algorithmen können verwendet werden, um Klartext auf elliptischen Kurven zu codieren. Diese Algorithmen haben die Eigenschaft, dass sie mit einer kleinen Wahrscheinlichkeit scheitern, einen Punkt auf der elliptischen Kurve zu produzieren. Bei geeigneter Wahl von Parametern kann diese Fehlerwahrscheinlichkeit beliebig klein gemacht werden, beispielsweise auf einen Wert unter 2^{-30}.

Das folgende Verfahren geht auf NEAL KOBLITZ zurück (siehe [129, p. 179]). Sei

$$E : y^2 \equiv (x^3 + ax + b) \bmod p$$

die elliptische Kurve. Die Nachricht m – bereits als Zahl codiert – wird in die x-Koordinate eines Punktes eingebettet. Die Wahrscheinlichkeit ist jedoch etwa $1/2$, dass der Ausdruck

$$m^3 + am + b \bmod p$$

[5] Polynomial in $\log_2 p$.

ein Quadrat modulo p ist. Man fügt daher einige Bits an das Ende von m an und wählt diese so, dass man eine Zahl x erhält, damit $x^3 + ax + b$ ein Quadrat modulo p ist.

Präziser formuliert: Sei k eine große positive ganze Zahl, so dass die Fehlerwahrscheinlichkeit $1/2^k$ akzeptierbar ist, wenn versucht wird, den Klartext m als Punkt der elliptischen Kurve zu codieren. Wir nehmen an, dass m die Ungleichung

$$(m+1) \cdot k < p$$

erfüllt. Die Nachricht wird als Zahl

$$x = mk + j \qquad 0 \leq j < k$$

dargestellt. Für $j = 0, 1, 2, \ldots, k-1$ berechnet man $x^3 + ax + b$ und versucht die Quadratwurzel mod p zu berechnen. Findet man einen solchen Wert y, dann setzt man

$$P_m = (x, y)$$

andernfalls wird j um 1 erhöht und man versucht mit dem neuen x die Berechnung der Quadratwurzel mod p. Dieses Verfahren wird solange wiederholt, bis entweder eine Wurzel gefunden ist oder $j = k$. Falls j den Wert k annimmt, ist der Versuch gescheitert, die Nachricht m als Punkt der elliptischen Kurve zu codieren. Da die Wahrscheinlichkeit, dass

$$x^3 + ax + b \bmod p$$

ein Quadrat ist, etwa $1/2$ ist, ist die Wahrscheinlichkeit bei dem Versuch zu scheitern, m als Punkt der elliptischen Kurve E darzustellen bei $1/2^k$.

Um die Nachricht aus dem Punkt $P_m = (x, y)$ zu dekodieren, berechnet man m einfach durch:

$$m = \lfloor \frac{x}{k} \rfloor$$

wobei $\lfloor \frac{x}{k} \rfloor$ die größte ganze Zahl kleiner oder oder gleich x/k bezeichnet.

Beispiel:

Wir betrachten die Primzahl $p = 179$ und die elliptische Kurve

$$E: \; y^2 \equiv x^3 + 2x + 7 \bmod 179.$$

Wir akzeptieren eine Fehlerwahrscheinlichkeit von $1/2^{10}$, daher können wir $k = 10$ setzen. Da die Wahl der Parameter die Bedingung

$$m \cdot k + k < p = 179$$

erfüllen muß, muß m die Bedingung

$$0 \leq m \leq 16$$

erfüllen. Wir nehmen daher an, die Nachricht ist $m = 5$. Daher haben wir x Werte der Form

$$x = m \cdot k + j = 50 + j$$

zu untersuchen. Die möglichen Werte von x sind daher $x = 50, 51, 52, \ldots, 59$.

Für $x = 50$ erhalten wir:

$$x^3 + 2x + 7 \bmod 179 \equiv 125.107 \bmod 179 \equiv 165 \bmod 179.$$

Für $x = 51$ erhalten wir:

$$x^3 + 2x + 7 \bmod 179 \equiv 132.760 \bmod 179 \equiv 121 \bmod 179.$$

Da

$$11^2 \bmod 179 \equiv 121 \bmod 179$$

können wir die Nachricht $m = 5$ als Punkt

$$P_{m=5} = (51, 11)$$

codieren. Die Nachricht m kann rekonstruiert werden über:

$$m = \lfloor \frac{x}{k} \rfloor = \lfloor \frac{51}{10} \rfloor = 5$$

11.5.2 Elliptische Kurven ElGamal Kryptosysteme

Wir rekapitulieren kurz die wesentlichen Punkte der nicht-elliptischen Kurvenversion des ElGamal Kryptosystems:

Alice möchte eine Nachricht m an Bob senden. Dazu wählt Bob eine Primzahl p und eine primitive Wurzel g modulo p. Zusätzlich wählt er eine geheime Zahl x und berechnet $y = g^x \bmod p$. Bob publiziert das Zahlentripel (Public Key) $KU = [p, g, y]$ und hält seinen privaten Schlüssel $KR = [x]$ geheim.

Alice wählt eine zufällige Zahl k und berechnet

$$y_1 = g^k \bmod p$$
$$y_2 = my^k \bmod p$$

Alice sendet das Chiffrat $C = (y_1, y_2)$ an Bob. Bob dechiffriert gemäß:

$$m \equiv (y_2 \cdot y_1^{-x}) \bmod p$$

Die Übertragung des ElGamal Kryptosystems auf elliptische Kurven lautet folgendermaßen (siehe [107, p. 14] oder [215, Chapt. 15.5]):

Szenario: Alice möchte Bob eine mit dem ElGamal Kryptosystem auf elliptischen Kurven verschlüsselte Nachricht übermitteln.

① Bob wählt eine elliptische Kurve $E \bmod p$, wobei p eine große Primzahl ist.

② Bob wählt einen Punkt α auf E und eine geheime Zahl x. Er berechnet

$$y = x \cdot \alpha = \underbrace{\alpha + \alpha + \ldots + \alpha}_{x \text{ mal}}$$

③ Bob hat damit den öffentlichen Schlüssel (Public Key) $\mathrm{K}_{pub}^{B} = [E, \alpha, y]$ und den Private Key $\mathrm{K}_{priv}^{B} = [x]$ erzeugt.

④ Alice besorgt sich den authentischen öffentlichen Schlüssel $\mathrm{K}_{pub}^{B} = [E, \alpha, y]$ von Bob.

⑤ Alice codiert ihre Nachricht m als Punkt P_m auf der elliptischen Kurve E.

⑥ Alice wählt eine zufällige Zahl k.

⑦ Alice berechnet die beiden Größen:

$$z_1 = k \cdot \alpha$$
$$z_2 = m + ky$$

⑧ Alice sendet das Chiffrat $C = (z_1, z_2)$ an Bob.

⑨ Bob dechiffriert durch die Berechnung

$$P_m = z_2 - xz_1$$

Beispiel: (siehe [218, pp. 251-253])

① Bob wählt die Primzahl $p = 11$ und die elliptische Kurve

$$E : y^2 = x^3 + 3x + 9$$

über $\mathbb{Z}_{11}$.

② Bob wählt einen Punkt auf der elliptischen Kurve, z.B.

$$\alpha = P = (2, 1)$$

$P \in E$, denn für $x = 2$ erhalten wir:

$$x^3 + 3x + 9 \bmod 11 = 23 \bmod 11 \equiv 1 \bmod 11$$

mit Wurzel $y = 1$.

③ Bob wählt nun eine geheime Zahl $x = 7$, die nur ihm bekannt ist. Bob berechnet nun

$$y = x \cdot P$$

Bob erhält den Wert $y = (3, 10)$.

Nebenrechnung:

Bob muß sukzessive die Reihe $2 \cdot P, 3 \cdot P, \ldots 7 \cdot P$ berechnen. Dies führt er folgendermaßen aus:

(a) $R = 2P$ mit

$$x_1 = x_2 = 2, y_1 = y_2 = 1$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_2 - x_3) - y_1 \end{aligned}$$

und

$$\lambda = (3x_1^2 + a)(2y_1)^{-1}$$

da $P = Q$. Es folgt:

$$\begin{aligned} \lambda &= (3x_1^2 + 3)(y_1)^{-1} \\ &= (3 \cdot 4 + 3) \cdot 2^{-1} \\ &\equiv 15 \cdot 6 \bmod 11 \\ &\equiv 2 \bmod 11 \end{aligned}$$

Damit ist

$$\begin{aligned} x_3 &= 4 - 4 = 0 \\ y_3 &= 2 \cdot 2 - 1 = 3 \end{aligned}$$

i.e.

$$2P = (0, 3)$$

(b) $R = 3P = (2, 1) + (0, 3)$ mit:

$$\begin{aligned} x_1 &= 2; \quad y_1 = 1 \\ x_2 &= 0; \quad y_2 = 3 \end{aligned}$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

und

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$

denn $P \neq Q$. Es folgt:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (3 - 1) \cdot (0 - 2)^{-1} \\ &= 2 \cdot (-2)^{-1} \\ &= 2 \cdot 9^{-1} \\ &= 2 \cdot 5 \\ &= 10 \end{aligned}$$

und:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 100 - 2 \\ &= 98 \equiv 10 \bmod 11 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= 10(2 - 10) - 1 \\ &= 10 \cdot 3 - 1 \\ &\equiv 7 \bmod 11 \end{aligned}$$

Damit:

$$3P = (10, 7)$$

(c) $R = 4P = (2, 1) + (10, 7)$ mit:

$$\begin{aligned} x_1 = 2; \quad y_1 = 1 \\ x_2 = 10; \quad y_2 = 7 \end{aligned}$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

sowie

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$

denn $P \neq Q$. Es folgt:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (7 - 1) \cdot (10 - 2)^{-1} \\ &= 6 \cdot (8)^{-1} \\ &= 6 \cdot 7 \\ &\equiv 9 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 81 - 2 - 10 \\ &= 69 \equiv 3 \bmod 11 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= 9(2 - 3) - 1 \\ &= 9 \cdot (-1) - 1 \\ &= 9 \cdot 10 - 1 \\ &\equiv 1 \bmod 11 \end{aligned}$$

Daraus folgt:

$$4P = (3, 1)$$

(d) $R = 5P = (2,1) + (3,1)$ mit:

$$\begin{aligned} x_1 &= 2; \quad y_1 = 1 \\ x_2 &= 3; \quad y_2 = 1 \end{aligned}$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

sowie

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$

denn $P \neq Q$. Es folgt:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= 0 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= -2 - 3 \\ &\equiv 6 \bmod 11 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= -1 \\ &\equiv 10 \bmod 11 \end{aligned}$$

Somit

$$5P = (6,10)$$

(e) $R = 6P = (2,1) + (6,10)$ mit:

$$\begin{aligned} x_1 &= 2; \quad y_1 = 1 \\ x_2 &= 6; \quad y_2 = 10 \end{aligned}$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

sowie

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$

denn $P \neq Q$. Es folgt:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (10 - 1) \cdot (6 - 2)^{-1} \\ &= 9 \cdot (4)^{-1} \\ &= 9 \cdot 3 \\ &\equiv 5 \bmod 11 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 25 - 2 - 6 \\ &= 17 \equiv 6 \bmod 11 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= 5(2-6) - 1 \\ &= 5 \cdot (-4) - 1 \\ &= 5 \cdot 7 - 1 \\ &\equiv 1 \bmod 11 \end{aligned}$$

Daraus folgt:

$$6P = (6, 1)$$

(f) $R = 7P = (2,1) + (6,1)$ mit:

$$\begin{aligned} x_1 = 2; \quad y_1 = 1 \\ x_2 = 6; \quad y_2 = 1 \end{aligned}$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

sowie

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$

denn $P \neq Q$. Es folgt:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &\equiv 0 \bmod 11 \end{aligned}$$

Damit:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= -2 - 6 \\ &= -8 \equiv 3 \bmod 11 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= -1 \\ &\equiv 10 \bmod 11 \end{aligned}$$

Daraus folgt:

$$7P = (3, 10)$$

④ Bob hat damit den öffentlichen Schlüssel (Public Key)

$$\mathrm{K}_{pub}^{B} = [E, \alpha, y]$$

mit

$$\begin{aligned} E &: y^2 \equiv x^3 + 3x + 9 \bmod 11 \\ \alpha &= (2,1) \\ y &= (3,10) \end{aligned}$$

und den Private Key $\mathrm{K}_{priv}^{B} = [x] = 7$ erzeugt.

⑤ Wir nehmen nun an, daß Alice die Nachricht

$$M = (10,4)$$

an Bob senden will. Die Nachricht ist bereits als Punkt der elliptischen Kurve codiert. Alice besorgt sich den Public-Key von Bob

$$\begin{aligned} E &: y^2 \equiv x^3 + 3x + 9 \bmod 11 \\ \alpha &= (2,1) \\ y &= (3,10) \end{aligned}$$

und wählt eine zufällige Zahl k mit $1 \leq k \leq 10$, zum Beispiel $k = 3$.

⑥ Alice berechnet mit Bobs öffentlichen Schlüssel und ihrem geheimen k die beiden Größen:

$$\begin{aligned} z_1 &= k \cdot \alpha \\ z_2 &= M + ky \end{aligned}$$

Sie erhält die Werte

$$\begin{aligned} z_1 &= 3 \cdot (2,1) \\ &= (10,7) \end{aligned}$$

und:

$$z_2 = (3,10)$$

Nebenrechnung:

Zu berechnen ist

$$z_2 = M + k \cdot y = (10,4) + 3 \cdot (3,10)$$

Zunächst ist $3 \cdot (3,10)$ zu berechnen; wir setzen:

$$x_1 = x_2 = 3, y_1 = y_2 = 10$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_2 - x_3) - y_1 \end{aligned}$$

mit der Steigung

$$\lambda = (3x_1^2 + a)(2y_1)^{-1}$$

da $P = Q$. Es folgt für die Steigung:

$$\begin{aligned}\lambda &= (3x_1^2 + 3)(2y_1)^{-1}\\ &= (3 \cdot 9 + 3) \cdot 20^{-1}\\ &= 30 \cdot 9^{-1}\\ &= 8 \cdot 5\\ &\equiv 7 \bmod 11\end{aligned}$$

Damit ist

$$\begin{aligned}x_3 &= 49 - 6 = 43 \equiv 10 \bmod 11\\ y_3 &= 7(3 - 10) - 10 \equiv 7 \bmod 11\end{aligned}$$

i.e.

$$2 \cdot (3, 10) = (10, 7)$$

Damit ist $3P = (3, 10) + (10, 7)$ mit:

$$\begin{aligned}x_1 = 3; \quad y_1 = 10\\ x_2 = 10; \quad y_2 = 7\end{aligned}$$

und

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2\\ y_3 &= \lambda(x_1 - x_3) - y_1\end{aligned}$$

und der Steigung

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$

denn $P \neq Q$. Es folgt:

$$\begin{aligned}\lambda &= (y_2 - y_1)(x_2 - x_1)^{-1}\\ &= (7 - 10) \cdot (10 - 3)^{-1}\\ &= (-3) \cdot (7)^{-1}\\ &= 8 \cdot 8\\ &= 64\\ &\equiv 9 \bmod 11\end{aligned}$$

und:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2\\ &= 81 - 3 - 10\\ &equiv2 \bmod 11\\ y_3 &= \lambda(x_1 - x_3) - y_1\\ &= 9(3 - 3) - 10\\ &= 10 \bmod 11\end{aligned}$$

Damit:

$$3P = (2, 10)$$

Schließlich ist noch zu berechnen:

$$z = (10, 4) + (2, 10)$$

$$\begin{aligned} x_1 &= 10; \quad y_1 = 4 \\ x_2 &= 2; \quad y_2 = 10 \end{aligned}$$

und

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

und der Steigung

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$

denn $P \neq Q$. Es folgt:

$$\begin{aligned} \lambda &= (y_2 - y_1)(x_2 - x_1)^{-1} \\ &= (10 - 4) \cdot (2 - 10)^{-1} \\ &= 6 \cdot 3^{-1} \\ &\equiv 6 \cdot 4 \bmod 11 \\ &\equiv 2 \bmod 11 \end{aligned}$$

und:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ &= 4 - 12 \\ &= -8 \\ &equiv3 \bmod 11 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ &= 2(10 - 3) - 4 \\ &= 10 \bmod 11 \end{aligned}$$

Damit:

$$z_2 = M + k\alpha = (3, 10)$$

Damit hat Alice den Chiffretext erstellt, dieser lautet:

$$C = (z_1, z_2) = \big((10, 7), (3, 10)\big)$$

⑦ Bob empfängt den von Alice erstellt Chiffretext C. Gemäß Algorithmus dechiffriert er das Chiffrat durch die Berechnung

$$P_m = z_2 - xz_1$$

Er berechnet zunächst

$$\begin{aligned} -xz_1 &= -7 \cdot (10, 7) \\ &= -(2, 10) \\ &= (2, -10) \\ &= (2, 1) \end{aligned}$$

Damit erhält Bob:

$$\begin{aligned} &P_m z_2 - xz_1 \\ &\quad = (3, 10) + (2, 1) \\ &\quad = (10, 4) \end{aligned}$$

11.5.3 Elliptische Kurven Diffie-Hellman Key Exchange

Die Übertragung des DIFFIE-HELLMAN Key Exchange Verfahrens lautet folgendermaßen:

Algorithmus Elliptic Curve Diffie-Hellman (ECDH)

① Szenario: Alice und Bob wollen einen Session Key mittels Elliptische Kurven Diffie-Hellman Key Exchange Verfahren austauschen.

② Alice und Bob einigen sich öffentlich auf eine elliptische Kurve E und einen Basispunkt $P \in E$.

③ Alice wählt eine zufällige Zahl γ_A, die sie geheim hält und berechnet

$$Y_A = \gamma_A \cdot P$$

Der Wert Y_A wird an Bob gesendet.

④ Bob wählt eine geheime zufällige Zahl γ_B und berechnet

$$Y_B = \gamma_B \cdot P$$

Der Wert Y_B wird an Alice gesendet.

⑤ Alice berechnet:

$$K = \gamma_A \cdot Y_B = \gamma_A(\gamma_B \cdot P)$$

⑥ Bob berechnet mit seinem Private Key γ_B:

$$K = \gamma_B \cdot Y_A = \gamma_B(\gamma_A \cdot P)$$

Beispiel:

❶ Alice und Bob einigen sich auf die elliptische Kurve

$$E : y^2 \equiv x^3 + x + 6 \bmod 11$$

über $\mathbb{Z}_{11}^*$, die in Abschnitt [11.3] explizit diskutiert ist. Als Basispunkt P wählen sie $P = (10, 2)$.

❷ Alice wählt eine geheime Zahl

$$\gamma_A = 3$$

und berechnet

$$\begin{aligned} Y_A &= \gamma_A \cdot P \\ &= 3 \cdot (10, 2) \\ &= (2, 7) \end{aligned}$$

Der Wert $Y_A = (2, 7)$ wird an Bob gesendet.

❸ Bob wählt eine geheime Zahl

$$\gamma_B = 8$$

und berechnet

$$\begin{aligned} Y_B &= \gamma_B \cdot P \\ &= 8 \cdot (10, 2) \\ &= (8, 8) \end{aligned}$$

Der Wert $Y_B = (8, 8)$ wird an Alice gesendet.

❹ Alice berechnet nun:

$$\begin{aligned} K &= \gamma_A \cdot Y_B \\ &= 3 \cdot (8, 8) \\ &= (3, 5) \end{aligned}$$

❺ Bob berechnet:

$$\begin{aligned} K &= \gamma_B \cdot Y_A \\ &= 8 \cdot (2, 7) \\ &= (3, 5) \end{aligned}$$

11.5.4 ElGamal digitale Signatur

Kapitel 12

Quantenkryptographie

Quantencomputing und Quantenkryptographie sind neue, interdisziplinäre Forschungsgebiete, die bereits erste praktische Anwendungen geliefert haben. Mittlerweile gibt es ausgezeichnete Darstellungen des Quanten Computings und der Quantenkryptographie, siehe z.B. [229], [215, Chap. 17], die Monographie von MICHAEL A. NIELSEN und ISAAC L. CHUANG [160].

Einige Meilensteine der Entwicklung klassischer und Quantencomputer sind:

1936 Die moderne Entwicklung der Computer Wissenschaft begann mit dem Jahr 1936 mit einer Publikation des englischen Mathematikers ALAN TURING mit dem Titel[1]:

On computable numbers with an Application to the Entscheidungsproblem.

In dieser Arbeit wird im Detail ein abstraktes Konzept dessen entwickelt, was man heute einen klassischen programmierbaren Computer nennt (siehe dazu [167]). Dieses mathematische Modell eines Computers nennt man (zu TURINGs Ehren) **Turing Maschine**. TURING zeigte auch, dass es eine **universelle** TURING Maschine gibt, die andere TURING Maschinen simulieren kann.

Weiterhin behauptete TURING, dass eine universelle TURING Maschine genau das vollständig beschreiben kann, was bei Ablauf der algorithmischen Abarbeitung eine Aufgabe geschieht. Das bedeutet also:

> Falls ein Algorithmus auf irgendeiner Hardware (e.g. ein moderner Computer) abgearbeitet werden kann, dann gibt es eine äquivalenten Algorithmus auf eine universellen TURING Maschine, der exakt die gleiche Aufgabe abrabeitet wie der Algorithmus auf dem Personal Computer.

Church-Turing These

[1] Alan M. Turing, *On computebale numbers with an Application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, 2nd series Vol 42 (1936), pp. 230 – 265

Der intuitive Algorithmusbegriff ist gleichbedeutend mit dem Algorithmusbegriff auf universellen TURING Machinen.

1945 Entwicklung der Grundprinzipien der universellen Rechenmaschinen durch JOHN VON NEUMANN und anderen, die VON NEUMANN Architektur.

1946/47 Bau des EDVAC, der erste nach dem VON NEUMANN Prinzip arbeitende Rechner.

1947 Entwicklung des Transistors durch JOHN BARDEEN, WALTER BRATTAIN und WILLIAM SHOCKLEY.

1965 GORDON MOORE formuliert das **Mooresche Gesetz**, nach dem sich die Anzahl der Transistoren pro Chip alle 18 Monate verdoppelt.

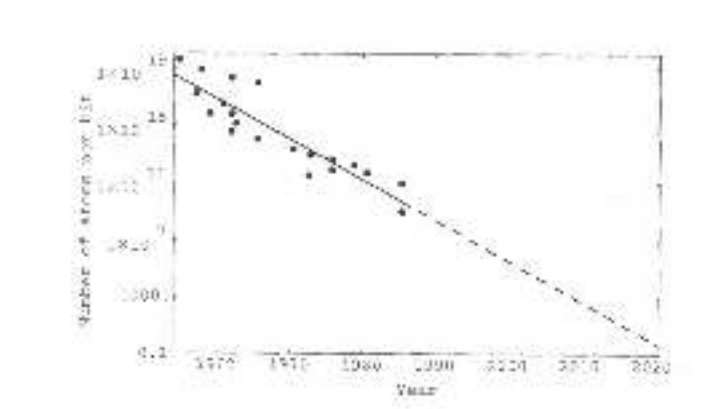


Abbildung 12.1: Anzahl der erforderlichen Atome um ein Bit darzustellen als Funktion des Kalenderjahres. Extrapolation dieses Trends läßt erwarten, dass um das Jahr 2020 das Ein-Atom-pro-Bit Level erreicht wird.

1961/1973 ROLF LANDAUER (IBM) und später CHARLES H. BENNETT untersuchen die Fragen:

- wo liegt die theoretisch untere Grenze für die Größe eines elektronischen Bauteils?
- wieviel Energie verbraucht ein Rechenvorgang mindestens?

1982 Der Physiknobelpreisträger RICHARD FEYNMAN formuliert die Vermutung, dass es prinzipielle Probleme geben kann, wenn man mit einem klassischen Computer versucht, quantenmechanische physikalische Systeme zu simulieren. FEYNMAN vermutete umgekehrt, wenn man aber Computer verwendet, die auf quantenmechanischen Prinzipen statt klassischen basieren, können Probleme berechnet werden, die auf klassischen Computern nicht umgesetzt werden können.

1982 Paul Benioff zeigt auf der Grundlage der Arbeiten von Landauer und Bennett, dass ein Computer, der ausschließlich nach den Gesetzen der Quantenmechanik arbeitet, theoretisch funktionieren kann.

1984 Charles Bennett von IBM und Gilles Brassard von der Universität Montreal entwickeln ein kryptographisches Protokoll ([18]), basierend auf quantenmechanischen Prinzipien, um einen sicheren (*i.e.* nicht kompromittierbaren) Schlüsselaustausch zwischen Alice und Bob zu ermöglichen. Dieses Protokoll ist unter dem Namen BB84 bekannt. Dieses Protokoll wurde bereits fünf Jahre später erstmals experimentell realisiert.

1985 Der britische Physiker David Deutsch untersucht die quantenmechanische Erweiterung des Konzepts der klassischen universellen Turing Maschine.

1989 Charles Bennett und vier Mitarbeiter erstellen bei IBM den ersten funktionsfähigen Prototypen eines Quantenkryptosystems [19].

1994 Peter W. Shor von den AT&T Research Labs zeigt, dass auf einem Quantencomputer die Faktorisierung ganzer Zahlen und das diskrete Logarithmusproblem effektiv berechnet werden können [201].

1995 Lov Grover zeigt, dass mit Quantencomputern das Suchproblem mit unstrukturiertem Datenmaterial (i.e. ein Beispiel ist die Telefonbuch Suche, bei der man die Telefonnummer kennt und den zugehörigen Namen sucht) effektiver durchgeführt werden kann als mit klassischen Computern.

2002 Einer Arbeitsgruppe der Universität Genf gelingt die *quantum key distribution* mit eine Plug&Play System auf einer Teststrecke von Genf nach Lausanne (67 km).

2004 In Wien wird unter Leitung von Anton Zeilinger die weltweit erste Demonstration einer abhörsicheren Banküberweisung mittels Quantenkryptographie durchgeführt.

2006 Der Genfer Arbeitsgruppe gelingt die erste Quantenteleportation über das Swisscom Telekommunikations Netzwerk.

2007 Anläßlich der Abstimmung über den Nationalrat in der Schweiz verbinden in Genf Wissenschaftler der örtlichen Universität erstmals zwei Wahlzentren über Glasfaser miteinander. Die darüber laufenden Daten sind mit einem quantenkryptographischen Verfahren verschlüsselt.

12.1 Ein Experiment mit Quanten

Quantenmechanik ist ungewöhnlich, da makroskopische Erfahrungen nicht anwendbar sind. Die Größenordnung, in der quantenmechanische Effekte und Phänomene auftreten, bewegen sich auf atomarer Ebene, was nicht ohne spezielles Equipment beobachtet werden kann. Dennoch gibt es einige Experimente, die quantenmechanische Effekte auf einer makroskopisch beobachtbaren Ebene zeigen.

Wir betrachten ein Experiment mit einer leistungsfähigen Lichtquelle, beispielsweise ein Laserpointer. Weiterhin verwenden wir drei Polarisationsfilter, die wir A, B und C nennen.

Die drei Polarisationsfilter werden in folgender Weise präpariert:

- Filter A polarisiert das durchgehende Licht horizontal
- Filter B polarisiert in einem Winkel von 45 Grad
- Filter C polarisiert vertikal.

Wir führen nun folgende Varianten eines Polarisationsexperiments durch:

1. Zunächst wird das Licht auf einen Schirm gestrahlt und zwischen Lichtquelle und Schirm wird der Polarisationsfilter A eingefügt (siehe Abbildung [12.2]).

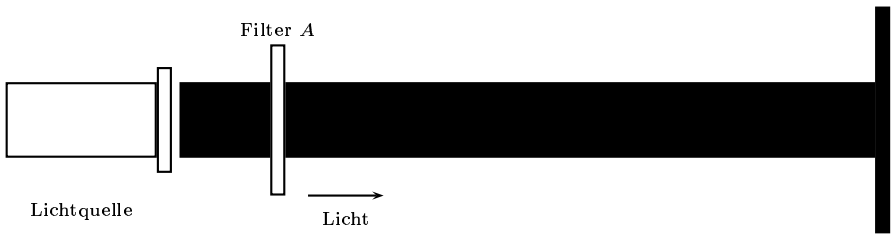


Abbildung 12.2: Das Polarisationsexperiment mit einem Filter A

Die Photonen, die den Filter passieren, haben horizontale Polarisation.

2. In einem zweiten Schritt fügen wir nun den Filter C in den Lichtstrahl ein wie in der Abbildung [12.3] angedeutet.

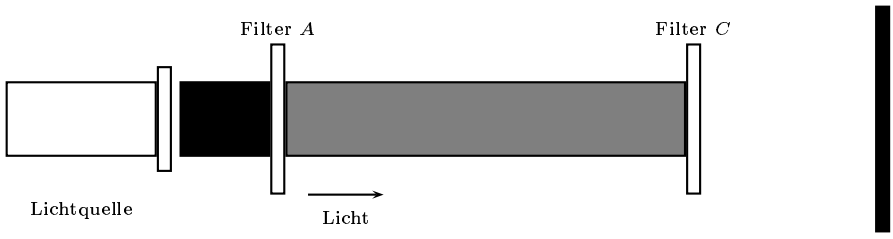


Abbildung 12.3: Das Polarisationsexperiment mit den beiden Filtern A und C

Da der Filter C vertikal polarisiert ist, filtert dieser alle horizontal polarisierten Photonen aus dem strahl, die durch den Filter A gehen. Daher erreicht kein Licht den hinter dem Filter C aufgestellten Schirm.

3. Der interessante Effekt stellt sich nun ein, wenn man den Filter B, der das Licht in einem Winkel von 45 Grad polarisiert, zwischen die beiden Filter A und C einfügt. Dieses Szenario ist in der Abbildung [12.4] skizziert.

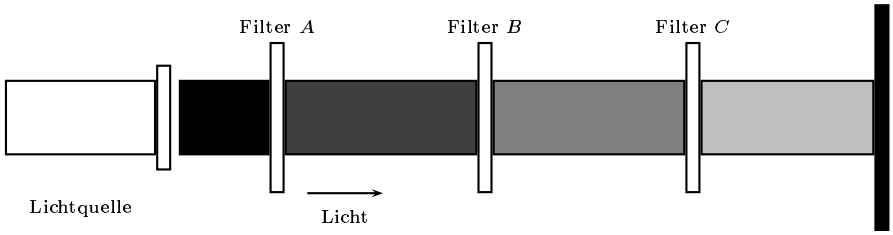


Abbildung 12.4: Das Polarisationsexperiment mit allen drei Filtern

Der ungewöhnliche Effekt ist, dass nun auf dem Schirm Licht ankommt, obwohl ja im Experiment 2 gezeigt ist, dass die beiden Filter A und C ausreichen, sämtliches licht zu absorbieren. Ein zusätzlicher Filter hat also den Effekt, dass Licht den Schirm erreicht.

Um dieses Experiment zu erklären, müssen wir etwas genauer betrachten, was Polarisation von Licht eigentlich bedeutet.

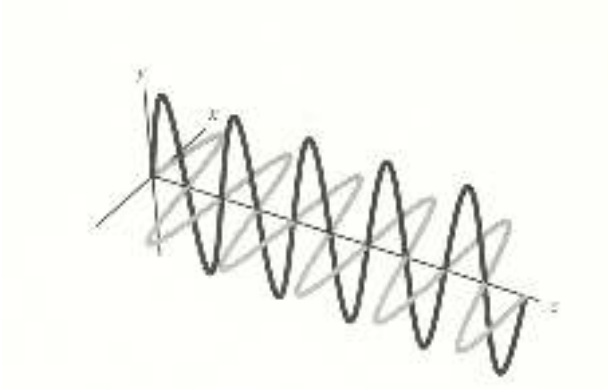


Abbildung 12.5: Aufbau einer elektromagnetischen Welle

Photonen sind elektromagnetische Wellen. man kann sich diese als oszillierendes elektrisches und oszillierendes magnetisches Feld vorstellen. Das elektrische und

magnetische Feld schwingen in Ebenen, die senkrecht zueinander und die auch noch senkrecht zur Fortpflanzungsrichtung des Photons stehen (siehe Abbildung [12.5]). In einem dreidimensionalen Koordinatensystem können wir uns also eine elektromagnetische Welle vorstellen, die sich in positiver z-Richtung fortpflanzt, das elektrische Feld schwingt dann in der $x-z$-Ebene, das magnetische Feld in der $y-z$-Ebene.

Eine Eigenschaft eines Photons besteht darin, dass der *Polarisationszustand* eines Photons genutzt werden kann, ein Bit zu codieren. Die uns interessierende Eigenschaft eines Photons ist also die Polarisation. Photonen können linear oder zirkular polarisiert sein. Lineare Polarisation bedeutet, wenn das Photon (oder die Lichtwelle) propagiert, dann bleibt das elektrische Feld in ein und derselben Schwingungsebene. bei zirkular polarisiertem Licht rotiert die Schwingungsebene des elektrischen Feldes während der Fortpflanzung des Lichts mit einer bestimmten Frequenz. Quantenkryptographie kann sowohl mit linear als auch zirkular polarisiertem Licht implementiert werden. Aus Gründen der Einfachheit beschränken wir uns hier auf linear polarisiertes Licht. Um ein Bit in einer elektromagnetischen Well zu codieren, ist es notwendig, ein Photon in einen speziellen Polarisationszustand zu setzen. Dies erfordert also, ein Photon zu erzeugen, dessen elektisches Feld in einer festgelegten Ebene schwingt. Eine Möglichkeit, dies zu erhalten ist, das Photon einfach durch einen Polarisationsfilter zu schicken, dessen Polarisationsachse auf den gewünschten Winkel gedreht ist.

Für die in der Abbildung [12.5] gezeigte Konfiguration –

- Fortpflanzungsrichtung der Welle ist die positive z-Richtung
- das elektrische Feld schwingt in der $x-z$-Ebene
- das nagnetische Feld schwingt in der $y-z$-Ebene

sagt man (per Konvention), das Licht ist *vertikal* polarisiert.

Wir stellen die Polarisation eines Photons durch einen Einheitsvektor im zweidimensioanlen (komplexen) Vektorraum dar. Eine geeignete Basis dieses Vektorraums bezeichnen wir mit $|\uparrow>$ und $|\rightarrow>$. Der Einheitsvektor $|\uparrow>$ beschreibt beispielsweise das vertikal polarisierte Photon, der Einheitsvektor $|\rightarrow>$ stellt das horizontal polarisierte Photon dar. Wir benutzen hier die in der Quantenmechanik übliche bracket-Notation $|\ \ >$ zur beschreibung eines quantenmechanischen Zustands. Ein beliebig polarisiertes Photon beschreibt man durch die linearkombination

$$\psi = \alpha_1 \,|\uparrow> + \alpha_2 \,|\rightarrow>,$$

mit komplexen Koeffizient, die die Bedingung

$$|\,\alpha_1\,|^2 + |\,\alpha_2\,|^2 = 1$$

erfüllen. Wir hätten auch eine andere orthogonale Basis des Vektorraums wählen können, beispielsweise die Zustände $|\nwarrow>$ und $|\nearrow>$.

Die Polarisationsfilter im obigen Experiment führen eine **Messung** des Polarisationszustandes der Photonen durch. Es gibt zwei mögliche Ausgänge: Entweder

ist die Polarisation des Photon gleichgerichtet mit der Polarisationsrichtung des Filters oder es ist senkrecht dazu polarisiert. Wird der (überlagerte) Zustand

$$\psi = \alpha_1 \mid\uparrow> + \alpha_2 \mid\rightarrow>,$$

durch einen vertkalen Polarisationsfilter gemessen, dann ist die Wahrscheinleichkeit, dass das Photon nach Passieren des Filters vertikal polarisiert ist $\mid \alpha_1 \mid^2$. Die Wahrscheinlichkeit, dass das photon horizontal polarisiert ist, beträgt $\mid \alpha_2 \mid^2$.

Nehmen wir nun an, wir messen ein vertikal polarisiertes Photon mit eine 45 Grad Filter. Da

$$\mid\uparrow> = \frac{1}{\sqrt{2}} \mid\nwarrow> + \frac{1}{\sqrt{2}} \mid\nearrow>$$

beträgt die Wahrscheinlichkeit, dass das Photon den Filter passiert (was gleichbedeutend damit ist, dass das Photon 45 Grad polarisiert ist), 1/2. Analog ist die Wahrscheinlichkeit, dass das Photon den Filter nicht passiert 1/2.

Eines der Grundprinziepien der Quantenmechnik ist, dass eine Messung das Photon in einen definierten Zustand überführt. Nach der Messung geht der Zustand des Photons in das Resultat der Messung über. Messen wir daher ein Photon im Zustand

$$\psi = \alpha_1 \mid\uparrow> + \alpha_2 \mid\rightarrow>,$$

als $\mid\rightarrow>$, dann befindet sich das Photon von diesem Zeitpunkt an in diesem Zustand. Messen wir dieses Photon mit eine weiteren $\mid\rightarrow>$-Filter, dann werden wir immer beobachten, dass sich das Photon im $\mid\rightarrow>$-zustand befindet. Messen wir mit einen vertikalen Filter $\mid\uparrow>$, werden wir niemals das Photon in diesem Zustand $\mid\uparrow>$ beobachten.

Kommen wir nun zur Interpretation des obigen Experiments. Die Photonen aus der Lichtquelle werden mit einer zufälligen Polarisation ausgestrahlt, i.e. die Photonen befinden sich in einem Zustand $\alpha_1 \mid\uparrow> + \alpha_2 \mid\rightarrow>$. Nur die Hälfte dieser Photonen passieren den Filter A, der horizontal polarisiert, und laufen auf den Schirm. Das heißt, die auf dem Schirm auftreffenden Photonen befinden sich alle im Zustand $\mid\rightarrow>$. Die andere Hälfte der Photonen wird absorbiert oder reflektiert und befinden sich im Zustand $mid \uparrow>$.

Bringt man nun den vertikal polarisierenden Filder C in diesen (horizontal polarisierten) Photonenstrahl, dann werden die Photonen sämtlich absorbiert und auf dem Schirm treffen keine Photonen auf.

Fügt man nun in der dritten Variante des Experiments den Filter B, der $\mid\nearrow>$ polarisiert, zwischen die Filter A und C, dann korrespondiert dies mit einer Messung des Zustandes $\mid\nearrow>$. Die auftreffenden Photonen im $\mid\rightarrow>$-Zustand passieren diesen Filter und befinden sich anschließend mit Wahrscheinlichkeit 1/2 im $\mid\nearrow>$-Zustand. Daher hat man eine 1:4 Reduktion der Anzahl der Photonen, die durch den Filter B laufen. Anschließend laufen die $\mid\nearrow>$ Photonen durch den $\mid\uparrow>$ Filter (Filter C) und passieren diesen mit Wahrscheinlichkeit 1/2. Daher ist die Gesamtintensität des Lichts, das auf den Schirm trifft 1/8 der ursprünglichen Intensität.

12.2 Schlüsselaustausch mit Quantenkryptographie

Nachdem wir nun einige grundlegenden Ideen quantenmechanischer Systeme kennengelernt haben, können wir diese nutzen, um eine Technik zu beschreiben, die es ermöglicht, Bits über einen Quantenkanal zu verteilen. Mit Hilfe dieser Bits kann ein Schlüssel aufgebaut werden, der die geheime Kommunikation über einen klassischen Kommunikationskanal ermöglicht, oder mit diesen Bits kann ein anderes Geheimnis zwischen Alice und Bob etabliert werden[2].

Wir beginnen damit, zu definieren, was ein **Quantenbit** oder kurz **Qubit** ist. Wie im letzten Abschnitt betrachten wir einen zweidimensionalen komplexen Vektorraum. Wir wählen ein Paar orthogonaler Vektoren der Länge 1 und bezeichnen diese mit $\mid 0 >$ und $\mid 1 >$. Beispielsweise können diese Vektoren den Zuständen $|\uparrow>$ und $|\rightarrow>$ oder dem Paar $|\nwarrow>$ und $|\nearrow>$ entsprechen. Ein Qubit ist ein Einheitsvektor in diesem Vektorraum. Für die Zwecke unserer aktuellen Diskussion können wir uns ein polarisiertes Photon als ein Qubit vortsellen. Wir haben die Notation $\mid 0 >$ und $\mid 1 >$ gewählt um die (klassischen) Bitzustände `0` bzw. `1` darzustellen. Die anderen Qubits sind einfach die Linearkombination dieser beiden Zustände $\mid 0 >$ und $\mid 1 >$.

Da ein Qubit ein Einheitsvektor ist, kann er in der Form

$$\text{qubit } = a \mid 0 > + b \mid 1 >$$

dargestellt werden, wobei a, b komplexe zahlen sind mit $\mid a \mid^2 + \mid b \mid^2 = 1$. Wie im fall der polarisierten Photonen können wir dieses Qubit bezüglich der Basis $\mid 0 >, \mid 1 >$ messen. Die Wahrscheinlichkeit, dass wir das Quabit im Zustand $\mid 0 >$ messen beträgt $\mid a \mid^2$.

Sehen wir uns nun an, wie Alice und Bob miteinander kommunizieren können, um eine geheime Nachricht auszutauschen. Sie benötigen dazu zwei Dinge:

- Einen Quantenkanal und
- einen klassischen Übertragungskanal.

Unter einem Quantenkanal versteht man einen Übertragungsweg, über den Alice und Bob polarisierte Photonen übertragne können, ohne dass Umgebungseinflüsse diese Photonen stören (typischerweise benutzt man dazu Glasfaser), i.e. es dürfen während der Übertragung der Photonen keine Wechselwirkungen der Photonen mit der Umwelt auftreten. Der kalssische Übertragungskanal wird zum Austausch der nachrichten genutzt. Wir nehmen an, dass die Angreiferin Eve beobachten kann, was über den klassischen Kanal übertragen wird und dass sie Photonen auf dem Quantenkanal beobachten und zurücksenden kann.

Alice beginnt den Aufbau einer Nachricht, indem sie eine Folge von Bits an Bob sendet. Diese Bits werden codiert durch eine zufällige Wahl einer Basis für jedes Bit. Dies geschieht folgendermaßen. Es gibt zwei Basen:

$$B_1 = \{|\uparrow>, |\rightarrow>\}, \quad B_2 = \{|\nwarrow>, |\nearrow>\}.$$

[2] Siehe [160, Chap. 12.6.3], [156, Chap. 6.2] oder [101, Chap. 6].

Wählt Alice die Basis B_1, dann codiert sie den Bitwert `0` durch $|\uparrow>$ und `1` durch $|\rightarrow>$. Wählt sie die Basis B_2, dann codiert sie `0` durch $|\nwarrow>$ und `1` durch $|\nearrow>$.
i

Jedesmal, wenn Alice ein Photon sendet, wählt Bob zufällig eine Basis B_1 oder B_2, um den Polarisationszuatnd des empfangenen Photons zu messen. Daher erhält er für jedes Photon als Resultat seiner Messung ein Element der gewählten Basis. Bob zeichnet diese Resultate seiner Messungen auf und hält diese Folge geheim. Anschließend teilt Bob Alice mit, welche Auswahl an Basen B_1 oder B_2 er bei jeder Messung getroffen hat. Alice wiederum teilt Bob ihrerseits mit, welche Basen für die Polarisation der Photonen die korrekten waren, die sie gesendet hat. Sie behalten die Bits, bei denen beide die gleichen Basen gewählt haben und verwerfen den Rest. Alice und Bob verfügen damit grob über die Hälfte der von Alice gesendeten Bits. Diese Bitfolge kann nun von Alice und Bob als geheimer Schlüssel für eine herkömmliche Verschlüsselung benutzt werden.

Beispiel:

Wir nehmen an, Alice möchte die Bits `0,1,1,1,0,0,1,0` an Bob senden. Zufällig wählt sie die Basenreihenfolge

$$B_1, B_2, B_1, B_1, B_2, B_2, B_1, B_2.$$

Daher sendet sie die folgenden Qubits über den Quantenkanal an Bob:

$$|\uparrow>, |\nearrow>, |\rightarrow>, |\rightarrow>, |\nwarrow>, |\nwarrow>, |\rightarrow>|\nwarrow> .$$

Bob seinerseits wählt (zufällig) die Basisreihenfolge

$$B_2, B_2, B_2, B_1, B_2, B_1, B_1, B_2.$$

Bob mißt die von Alice erhaltenen Qubits und teilt ihr mit, welche Reihenfolge der Basen er gewählt hat. Alice wiederum teilt Bob mit, dass seine 2., 4., 5., 7. und 8. Wahl mit ihrer übereinstimmt. Diese fünf Basen führen auf die folgenden Messungen auf Bobs Seite:

$$|\nearrow>, |\rightarrow>, |\nwarrow>, |\rightarrow>, |\nwarrow> .$$

Dies entspricht den (klassischen) Bits: `1,1,0,1,0`. Daher verfügen nun Alice und Bob über den gleichen Bitstring `1 1 0 1 0`, den sie beide als Input für den Schlüssel eines symmetrischen Kryptosystems benutzen können.

Die Sicherheit des Schlüsselaustauschs mit Hilfe quantenkryptographischer verfahren beruht auf den gesetzen der Quantenmechanik und dem fundamentalen Prinzip, dass eine Messung den Zustand eines Teilchens ändert. Da die Angreiferin Eve messungen an den Photonen vornehmen muß, um die Photonenübertragung zwischen Alice und Bob beobachten zu können, wird Eve durch diese Messungen Fehler in die Daten einfügen, die zwischen Alice und Bob ausgetauscht werden.

Sehen wir uns an, wie dies geschieht.

Angenommen, Eve mißt die Zustände der von Alice gesendeten Photonen und es gelingt ihr, diese Photonen dann an Bob weiterzuleiten. Da diese Photonen von Alice gemessen werden, haben diese Photonen – wenn sie bei Bob ankommen – die Zustände, die Eve beobachtet hat. Wenn nun Bob seine Messungen durchführt und er verwendet die korrekte Basis (die er von Alice hat), dann hat er eine 25% Wahrscheinlichkeit, dass er den falschen Wert misst.

Sehen wir uns diesen Punkt etwas genauer an. Angenommen, Alice sendet ein horizontal polarisiertes Photon im Zustand $|\rightarrow>$ und Bob benutzt die gleiche Basis B_1 wie Alice. Nun können zwei Fälle eintreten:

1. Eve benutzt die gleiche Basis wie Alice und Bob, nämlich ebenfalls B_1. Dann geht das Photon unverändert durch Eves Messung und Bob seinerseits misst den korrekten Wert.

2. Eve wählt die Basis $B_2 = \{|\nwarrow>, |\nearrow>\}$. In diesem fall misst Eve mit gleicher Wahrscheinlichkeit die Zustände $|\nwarrow>$ und $|\nearrow>$. Die Photonen, die zu Bob gelangen, werden in einem dieser Zustände sein und er wird daher in der Hälfte der Fälle den korrekten Wert $|\rightarrow>$ und in der anderen Hälfte den falschen Wert messen.

Kombiniert man also die beiden möglichen Auswahlen an Basen, die Eve hat, dann hat das zur Folge, dass Bob mit 25%iger Wahrscheinlichkeit den inkorrekten Wert zu messen.

Daher erhöht das Abhören eines Quantenkanals die Fehlerrate in der Kommunikation zwischen Alice und Bob. Überprüfen nun Alice und Bob ihre daten auf Diskrepanzen über den klassischen Kommunikationskanal, werden sie den Lauschangriff bemerken.

Teil II

Mathematische Grundlagen

Kapitel 13

Algebraische Strukturen

Die grundlegenden mathematischen Werkzeuge, die in der Kryptologie zum Einsatz kommen, stammen aus der mathematischen Disziplin der Zahlentheorie. Dies ist (für Nicht–Mathematiker) ein etwas exotischer Bereich, daher beginnen wir erst mal mit der Festlegung der Notation.[1]

In der elementaren Algebra, die man in der Schule lernt, verwendet man Operationen der Arithmetik, wie beispielsweise Addition und Multiplikation, wobei spezielle Zahlen jedoch durch Symbole (Variablen) ersetzt werden. Ziel dabei ist, allgemeingültige Formeln abzuleiten. Werden dann in diesen Formeln die Variablen durch konkrete Zahlen ersetzt, erhält man Lösungen spezieller numerischer Probleme. In der modernen Algebra geht man noch einen Abstraktionsgrad weiter. Anstatt der Untersuchung der gewohnten arithmetischen Operationen auf reellen Zahlen, betrachtet man allgemeine Operationen — Prozesse um zwei oder mehrere Elemente zu kombinieren, um ein weiteres Element zu erhalten — auf allgemeinen Mengen.[2] Das Ziel dabei ist, gemeinsame Eigenschaften aller Systeme zu finden. Solche Systeme bestehen aus allgemeinen Mengen, auf denen eine feste Anzahl von Operationen definiert sind, die auf eine definitive Weise untereinander zusammenhängen. Beispielsweise untersucht man Mengen mit zwei binären Operationen, die sich wie die Addition und Multiplikation reeller Zahlen verhalten.

Wir bezeichnen die Menge der **natürlichen Zahlen** durch das Symbol $\mathbb{N}$, das heisst, natürliche Zahlen sind Elemente der Menge

$$\mathbb{N} = \{0, 1, 2, 3, \dots\}.$$

Die Zahl '0' ist ebenfalls Element dieser Menge.[3]

Um negative Zahlen betrachten zu können, muss die Menge der natürlichen

[1] Eine sehr ausführliche und gut lesbare Einführung in die hier betrachtete Thematik findet man in dem Buch von Pierre Basieux [175]. Siehe auch [232] für eine technische Einführung in die vorliegende Thematik. Die ultimative Referenz für endliche Körper ist das Buch von Lidl und Niederreiter [144].

[2] Also nicht nur Zahlenmengen.

[3] Die Notation wird in der Literatur nicht einheitlich gehandhabt, oftmals wird mit $\mathbb{N}$ auch die Menge der natürlichen Zahlen ohne die '0' bezeichnet und $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

Zahlen erweitert werden zu der Menge der **ganzen Zahlen**, die wir mit

$$\mathbb{Z} = \{\ldots, -4, -3, -2, -1, 0, 1, 2, 3, \ldots\}$$

bezeichnen. Die mathematische Teildisziplin der **Zahlentheorie** beschäftigt sich fast ausschließlich mit den Eigenschaften dieser Menge.

Eine **binäre Operation** $\circ$, die auf einer Menge $\mathbb{A}$ operiert, ist eine Abbildung der Form[4]

$$\begin{aligned} \circ : \mathbb{A} \times \mathbb{A} &\longrightarrow \mathbb{A} \\ a\,,\, b &\longmapsto \circ(a,b) = a \circ b, \end{aligned}$$

i.e. zwei Elemente der Menge $\mathbb{A}$ werden auf ein anderes Element dieser Menge abgebildet. Man sagt, die Menge $\mathbb{A}$ ist *abgeschlossen* unter der Verknüpfung $\circ$.

Unter einer **algebraischen Struktur** oder einem **algebraischen System** versteht man eine Menge zusammen mit einer oder mehreren Operationen auf der Menge.

13.1 Halbgruppen

Die Menge der natürlichen oder ganzen Zahlen zusammen mit einer der bekannten binären Operation, die auf diesen Mengen definiert ist — *e.g.* Addition, Subtraktion, usw. — bilden die Grundlage der sogenannten **algebraischen Strukturen**.

Definition [13.12]:

Eine **Halbgruppe** ist ein Paar $(\mathbb{A}, \circ)$, wobei $\mathbb{A}$ eine Menge und $\circ$ eine Abbildung

$$\circ : \mathbb{A} \times \mathbb{A} \longrightarrow \mathbb{A}$$

ist. Die Abbildung erfüllt das **Assoziativgesetz**

$$a \circ (b \circ c) = (a \circ b) \circ c \quad \forall\, a, b, c \in \mathbb{A}.$$

Beispiel [13.15]

☞ Die Paare $(\mathbb{N}, +)$ und $(\mathbb{N}, \cdot)$ sind Halbgruppen, da die Addition und Multiplikation natürlicher Zahlen assoziative Operationen sind und die Menge $\mathbb{N}$ unter diesen Operationen abgeschlossen sind.

[4] Das *kartesische Produkt* zweier Mengen $\mathbb{K}$ und $\mathbb{L}$, bezeichnet mit $\mathbb{K} \times \mathbb{L}$ ist die Menge der geordneten Paare (k, l) mit $k \in \mathbb{K}$ und $l \in \mathbb{L}$.

- ☞ Das Paar $(\mathbb{N}, -)$ hat nicht die Struktur einer Halbgruppe, denn die Subtraktion zweier natürlicher Zahlen ist im allgemeinen nicht wieder eine natürliche Zahl. Mit anderen Worten, die Menge $\mathbb{N}$ ist nicht abgeschlossen unter der Subtraktionsoperation.
- ☞ Aus dem gleichen Grund ist die Menge der natürlichen Zahlen mit der Divisionsoperation keine Halbgruppe, da die Division zweier natürlichen Zahlen in der Regel eine rationale Zahl ergibt.

Anmerkung:

Die in desem Kapitel betrachteten algebraischen Strukturen sind keinesfalls nur auf Zahlenmengen beschränkt.

Betrachte zum Beispiel die Menge der Worte über dem Alphabet $\mathbb{B} = \{0, 1\}$, *i.e.* die Menge aller Bitstrings. Diese Menge ist:

$$\begin{aligned}\mathbb{B}^* = {} & \lambda \cup \{0, 1\} \cup \{00, 01, 10, 11\} \cup \\ & \cup \{000, 001, 010, 011, 100, 101, 110, 111\} \cup \\ & \cup \ldots\end{aligned}$$

wobei λ das leere Wort ist. Worte über dem Alphabet $\mathbb{B}$ sind also Bitstrings beliebiger Länge. Über dieser Menge $\mathbb{B}^*$ betrachten wir die binäre Operation der *Konkatenation*, *i.e.* das Hintereinandersetzen von Worten aus $\mathbb{B}^*$. Diese Operation bildet Bitstrings der Länge n_1 und n_2 auf Bitstrings der Länge $n_1 + n_2$ ab. Die Menge der Worte über $\mathbb{B}$ ist daher abgeschlossen unter der Konkatenationsoperation.

$$\begin{aligned}| : \mathbb{B}^* \times \mathbb{B}^* &\longrightarrow \mathbb{B}^* \\ (w_1, w_2) &\longmapsto w_1 \mid w_2 = w_1 w_2 \in \mathbb{B}^*\end{aligned}$$

Es ist leicht einzusehen, dass diese Abbildung das Assoziativgesetz erfüllt, *i.e.*

$$w_1 \mid (w_2 \mid w_3) = (w_1 \mid w_2) \mid w_3.$$

Daher hat das Paar $(\mathbb{B}^*, |)$ die algebraische Struktur einer Halbgruppe.

Das Paar $(\mathbb{B}^*, |)$ hat sogar eine weitere Eigenschaft. Konkateniert man ein beliebiges Wort $w \in \mathbb{B}^*$ mit dem leeren Wort λ, wird das Wort w reproduziert:

$$w \mid \lambda = \lambda \mid w = w \quad \forall\, w \in \mathbb{B}^*$$

i.e. λ ist das **neutrale Element** der Konkatenation. Man nennt eine Halbgruppe mit einem neutralen Element ein **Monoid**.

13.2 Gruppen

Definition [13.13]:

Eine **Gruppe** ist eine Halbgruppe $(\mathbb{A}, \circ)$ mit den folgenden Eigenschaften:

1. Es gibt ein *neutrales Element* $\mathbf{1}_{\mathbb{A}} \in \mathbb{A}$ mit
$$a \circ \mathbf{1}_{\mathbb{A}} = a \qquad \forall\, a \in \mathbb{A}.$$
2. Für jedes $a \in \mathbb{A}$ existiert ein Element $a^{-1} \in \mathbb{A}$ — das *Inverse* von a — mit
$$a \circ a^{-1} = \mathbf{1}_{\mathbb{A}} \qquad \forall a \in \mathbb{A}.$$

Anmerkung:

Falls $(\mathbb{A}, \circ)$ eine Gruppe ist und die Operation $\circ$ ist kommutativ, *i.e.* es gilt:

$$a \circ b = b \circ a \qquad \forall\, a, b \in \mathbb{A},$$

dann nennt man die Gruppe auch **Abelsche Gruppe** nach dem norwegischen Mathematiker Niels Henrik Abel (1802 – 1829). oder kommutative Gruppe.

Man zeigt leicht mit den obigen Eigenschaften, dass das neutrale Element und das inverse Element a^{-1} eines gegebenen Elements $a \in \mathbb{A}$ eindeutig sind. Weiterhin gilt:

$$(a \circ b)^{-1} = b^{-1} \circ a^{-1} \quad \forall a, b \in \mathbb{A}.$$

Anmerkung:

Der Einfachheit halber verwenden wir gelegentlich das übliche Multiplikationszeichen als Schreibweise für die Verknüpfung $\circ$ der Gruppe, *i.e.* wir schreiben ab oder $a \cdot b$ oder $a \times b$ anstatt $a \circ b$. Es muss aber betont werden, dass dadurch keineswegs angenommen wird, dass die Operation die übliche Multiplikation von Zahlen ist. Manchmal ist es auch zweckmäßig, für $a \circ b$ die Schreibweise $a + b$ zu verwenden und $-a$ anstelle von a^{-1}.

Das Assoziativgesetz gewährleistet, dass Ausdrücke der Form

$$a_1 a_2 \ldots a_n \quad a_j \in \mathbb{A}, 1 \leq j \leq n$$

eindeutig sind. Für die n-fache Verknüpfung eines Elements $a \in \mathbb{A}$ mit sich selbst ($n \in \mathbb{N}$) schreiben wir im Fall der Multiplikationsnotation:

$$a^n = \underbrace{aa \ldots a}_{n \text{ Faktoren } a}.$$

Benutzt man die Additionsschreibweise für die Operation $\circ$ auf $\mathbb{A}$, dann schreiben wir

$$na = \underbrace{a + a + \ldots + a}_{n \text{ Summanden } a}.$$

Mit den gebräuchlichen Schreibweisen werden wir die folgenden Rechenregeln verwenden:

- Multiplikationsschreibweise:

$$\begin{aligned} a^{-n} &= (a^{-1})^n \\ a^m a^n &= a^{m+n} \\ (a^m)^n &= a^{mn} \end{aligned}$$

- Additive Notation:

$$\begin{aligned} (-n)a &= n(-a) \\ ma + na &= (m+n)a \\ m(na) &= (mn)a \end{aligned}$$

Für $n = 0 \in \mathbb{Z}$ benutzen wir die Konvention

$$a^0 = \mathbf{1}_{\mathbb{A}}$$

in der Multiplikationsschreibweise und

$$0a = \mathbf{1}_{\mathbb{A}}$$

in der additiven Notation.

13.2.1 Beispiele

Zahlenmengen, die mit einer binären Operation die Struktur einer Gruppe bilden sind:

- ☞ Die Menge der ganzen Zahlen mit der binären Operation Addition, *i.e.* $(\mathbb{Z}, +)$, ist eine Gruppe. Das neutrale Element dieser Gruppe ist das Element $0 \in \mathbb{Z}$, das inverse Element von $a \in \mathbb{Z}$ ist das Element $-a \in \mathbb{Z}$.
- ☞ Das Paar $(\mathbb{Z}, \cdot)$ hat nicht die algebraische Struktur einer Gruppe, da in $\mathbb{Z}$ kein inverses Element zu jedem $a \in \mathbb{Z}$ gefunden werden kann.
- ☞ Betrachtet man das Paar $(\mathbb{Q} \setminus \{0\}, \cdot)$, wobei $\mathbb{Q}$ die Menge der rationalen Zahlen ist, dann kann leicht verifiziert werden, dass dieses Paar die Gruppenstruktur hat. Das neutrale Element bezüglich der Multiplikation ist die 1, das Inverse von $a \in \mathbb{Q}$ ist $1/a \quad \forall a \in \mathbb{Q} \setminus \{0\}$.
- ☞ Sei G die Menge der Reste aller ganzen Zahlen bei der Division durch 6, *i.e.*

$$G = \{0, 1, 2, 3, 4, 5\}$$

und sei $a \circ b$ der Rest bei der Division der gewöhnlichen Summe von a und b. Die Existenz eines Elements für die Identität und ein Inverses ist offensichtlich. Die Assoziativität von $\circ$ erfordert einige Rechenarbeit. Diese Gruppe kann auf beliebige n verallgemeinert werden.

Beispiel [13.16]

Die Cäsar Chiffre

In diesem ausführlichen Beispiel betrachten die CÄSAR Chiffre (oder auch Shift Chiffre genannt) als Abbildung

$$\begin{aligned} \text{CAESAR}_K \; &: \mathbb{Z}_{26} \longrightarrow \mathbb{Z}_{26} \\ &\quad x \longmapsto y = (x + K) \bmod 26 \end{aligned}$$

Die Hintereinanderausführung zweier CÄSAR Verschlüsselungen ist:

$$\begin{aligned} \big(\text{CAESAR}_{K_1} \circ \text{CAESAR}_{K_2}\big)(x) &= \text{CAESAR}_{K_1}\big(\text{CAESAR}_{K_2}(x)\big) \\ &= \text{CAESAR}_{K_1}\big((x + K_2) \bmod 26\big) \\ &= \text{CAESAR}_{K_1}(y) \\ &= z \\ &= (y + K_1) \bmod 26 \\ &= \big[(x + K_2) \bmod 26 + K_1\big] \bmod 26 \\ &= \big((x + K_2) + K_1\big) \bmod 26 \\ &= \big(x + (K_2 + K_1)\big) \bmod 26 \\ &= \big(x + K_3\big) \bmod 26 \\ &= \text{CAESAR}_{K_3}(x) \end{aligned}$$

mit

$$K_3 = K_2 + K_1$$

oder:

$$\text{CAESAR}_{K_1} \circ \text{CAESAR}_{K_2} = \text{CAESAR}_{K_1 + K_2}$$

Bemerkungen:

❶ In obiger Notation ist $\circ$ (i.e. die Hintereinanderausführung zweier CÄSAR Verschlüsselungen) eine binäre Operation auf der Menge der CÄSAR Verschlüsselungen (Shift Chiffren) über $\mathbb{Z}_{26}$.

❷ In der obigen Ableitung der Hintereinanderausführung zweier CÄSAR Verschlüsselungen haben wir verschiedene Eigenschaften der `mod` Operation genutzt, die aus der *modularen Arithmetik* stammen (siehe dazu Abschnitt [15.3.1]).

❸ Damit haben wir gezeigt, daß die Hintereinanderausführung zweier CÄSAR Verschlüsselungen mit Parametern K_1 bzw. K_2 identisch ist zu einer einzigen CÄSAR Verschlüsselung mit Parameter $K_3 = K_1 + K_2$. Mit anderen Worten, die Menge der CÄSAR Verschlüsselungen ist abgeschlossen unter der Hintereinanderausführung.

❹ Algebraische Untersuchungen von Verschlüsselungen wie wir sie hier an einem sehr einfachen Beispiel durchgeführt haben, gehen zurück auf eine richtungsweisende Arbeit von CLAUDE SHANNON aus dem Jahre 1949 (siehe [199]).

❺ Aus der Sicht der Kryptographie bedeutet das oben abgeleitete Ergebnis, daß die zweimalige Verschlüsselung eines Klartextes mit der CÄSAR Verschlüsselung *keine* höhere Sicherheit bietet als eine einfache CÄSAR Verschlüsselung. Denn eine zweifache Verschlüsselung ist gleichwertig zu einer einfachen Verschlüsselung mit dem Schlüssel $K_3 = K_1 + K_2$.

Behauptung:

Das Paar (CAESAR, $\circ$), wobei $\circ$ die Hintereinanderausführung zweier CÄSAR Verschlüsselungen bedeutet, ist eine ABELsche Gruppe.

Beweis:

❶ Assoziativgesetz
Aufgrund der Assoziativität der Addition zeigt man leicht, dass gilt:

$$\left(\text{CAESAR}_{k_1} \circ \text{CAESAR}_{k_2}\right) \circ \text{CAESAR}_{k_3} = \text{CAESAR}_{k_1} \circ \left(\text{CAESAR}_{k_2} \circ \text{CAESAR}_{k_3}\right)$$

❷ Neutrales Element
Für jedes $x \in \mathbb{Z}_{26}$ gilt

$$\text{CAESAR}_{k=0}(x) = x$$

i.e.

$$\text{CAESAR}_k \circ \text{CAESAR}_0 = \text{CAESAR}_k$$

❸ Inverses Element
Man prüft leicht nach, daß gilt:

$$\text{CAESAR}_k \circ \text{CAESAR}_{(-k)} = \text{CAESAR}_0$$

Beispiel [13.17]

Die Affine Chiffre

In einem zweiten ausführlichen Beispiel untersuchen wir die **affine Chiffre** (siehe zum Beispiel [213, 215]):

$$\begin{aligned} A_{(\alpha,\beta)} &: \mathbb{Z}_{26} \longrightarrow \mathbb{Z}_{26} \\ x &\longmapsto y = (\alpha x + \beta) \bmod 26 \\ &\quad \text{mit } \alpha, \beta \in \mathbb{Z}_{26} \text{ und } \operatorname{ggT}(\alpha, 26) = 1 \end{aligned}$$

Die Hintereinanderausführung zweier affiner Verschlüsselungen ist folgendermaßen durchzuführen. Wir setzen:

$$y = A_{(\alpha,\beta)}(x) = (\alpha x + \beta) \bmod 26$$

Dann ist

$$\begin{aligned} \Big(A_{(\alpha_2,\beta_2)} \circ A_{(\alpha_1,\beta_1)}\Big)(x) &= A_{(\alpha_2,\beta_2)}\Big(A_{(\alpha_1,\beta_1)}(x)\Big) \\ &= A_{(\alpha_2,\beta_2)}(y) \\ &= (\alpha_2 y + \beta_2) \bmod 26 \\ &= \big[\alpha_2 \cdot \big((\alpha_1 x + \beta_1) \bmod 26\big) + \beta_2\big] \bmod 26 \\ &= \big(\alpha_2 \cdot \alpha_1 \cdot x + \alpha_2 \cdot \beta_1\big) \bmod 26 + \beta_2 \bmod 26 \\ &= \big(\alpha_2 \cdot \alpha_1 \cdot x + \alpha_2 \cdot \beta_1 + \beta_2\big) \bmod 26 \\ &= (\alpha_3 x + \beta_3) \bmod 26 \end{aligned}$$

mit

$$\begin{aligned} \alpha_3 &= \alpha_2 \cdot \alpha_1 \\ \beta_3 &= \alpha_2 \cdot \beta_1 + \beta_2 \end{aligned}$$

Die Hintereinanderausführung zweier affiner Chiffren ist daher genau dann wieder eine affine Chiffre mit Parametern (Schlüssel) (α_3, β_3), wenn gilt:

$$\operatorname{ggT}(\alpha_3, 26) = \operatorname{ggT}(\alpha_2 \cdot \alpha_1, 26) = 1.$$

Mit anderen Worten, es ist zu zeigen, wenn gilt

$$\operatorname{ggT}(\alpha_1, 26) = 1 \text{ und } \operatorname{ggT}(\alpha_1, 26) = 1$$

dann folgt hieraus

$$\operatorname{ggT}(\alpha_2 \cdot \alpha_1, 26) = 1.$$

Bevor wir dies zeigen, betrachten wir die Hintereinanderausführung zweier affiner Chiffren in umgekehrter Reihenfolge:

$$\begin{aligned} \Big(A_{(\alpha_1,\beta_1)} \circ A_{(\alpha_2,\beta_2)}\Big)(x) &= A_{(\alpha_1,\beta_1)}\Big(A_{(\alpha_2,\beta_2)}(x)\Big) \\ &= A_{(\alpha_1,\beta_1)}(y) \\ &= (\alpha_1 x + \beta_1) \bmod 26 \\ &= \big[\alpha_1 \cdot \big((\alpha_2 x + \beta_2) \bmod 26\big) + \beta_1\big] \bmod 26 \\ &= \big(\alpha_1 \cdot \alpha_2 \cdot x + \alpha_1 \cdot \beta_2\big) \bmod 26 + \beta_1 \bmod 26 \\ &= \big(\alpha_1 \cdot \alpha_2 \cdot x + \alpha_1 \cdot \beta_2 + \beta_1\big) \bmod 26 \\ &= (\alpha_3' x + \beta_3') \bmod 26 \end{aligned}$$

mit

$$\alpha_3' = \alpha_1 \cdot \alpha_2 = \alpha_3$$
$$\beta_3' = \alpha_1 \cdot \beta_2 + \beta_1 \neq \beta_3$$

Mit anderen Worten, die Hintereinanderausführung zweier affiner Chiffren ist nichtkommutativ.

Behauptung:

Wenn gilt:

$$\text{ggT}(\alpha_1, 26) = 1 \text{ und } \text{ggT}(\alpha_2, 26) = 1$$

dann folgt

$$\text{ggT}(\alpha_2 \cdot \alpha_1, 26) = 1.$$

Beweis:

Wir führen einen konstruktiven Beweis durch, um die obige Aussage zu zeigen. Zu zeigen ist:

$$\text{ggT}(\alpha_2 \cdot \alpha_1, 26) = 1$$

das bedeutet, es gibt Zahlen $x, y \in \mathbb{Z}$ mit:

$$x \cdot (\alpha_1 \cdot \alpha_2) + y \cdot 26 = 1$$

Laut Voraussetzung ist

$$\text{ggT}(\alpha_1, 26) = 1$$

i.e. es gibt Zahlen $x_1, y_1 \in \mathbb{Z}$ mit[5]

$$x_1 \cdot \alpha_1 + y_1 \cdot 26 = 1$$

Außerdem:

$$\text{ggT}(\alpha_2, 26) = 1$$

i.e. es gibt Zahlen $x_2, y_2 \in \mathbb{Z}$ mit

$$x_2 \cdot \alpha_2 + y_2 \cdot 26 = 1$$

Wir multiplizieren beide Gleichungen und erhalten:

$$\begin{aligned} 1 &= (\alpha_1 x_1 + 26 y_1) \cdot (\alpha_2 x_2 + 26 y_2) \\ &= \alpha_1 \alpha_2 x_1 x_2 + 26 \alpha_1 x_1 y_2 + 26 y_1 \alpha_2 x_2 + 26 \cdot 26 \cdot y_1 y_2 \\ &= (x_1 x_2) \cdot (\alpha_1 \cdot \alpha_2) + (\alpha_1 x_1 y_2 + \alpha_2 x_2 y_1 + 26 y_1 y_2) \cdot 26 \\ &= x_3 (\alpha_1 \cdot \alpha_2) + y_3 \cdot 26. \end{aligned}$$

Also, wir haben zwei Zahlen gefunden:

$$\begin{aligned} x_3 &= x_1 \cdot x_2 \\ y_3 &= \alpha_1 x_1 y_2 + \alpha_2 x_1 y_1 + 26 y_1 y_2 \end{aligned}$$

mit $x_3, y_3 \in \mathbb{Z}$, so dass

$$x_3 (\alpha_1 \cdot \alpha_2) + y_3 \cdot 26 = 1$$

[5] Dies ist das Lemma von BEZOUT, siehe Theorem [5].

Daraus folgt:

$$\text{ggT}(\alpha_2 \cdot \alpha_1, 26) = 1$$

Beispiel [13.18]

Das folgende Beispiel illustriert die Logik des obigen Beweises.

1. $\alpha_1 = 3$. Es gilt:

   $$\text{ggT}(3, 26) = 1$$

   Daher gibt es Zahlen $x_1, y_1 \in \mathbb{Z}$ mit

   $$x_1 \cdot 3 + y_1 \cdot 26 = 1$$

   Diese Gleichung wird durch

   $$x_1 = 9, y_1 = -1$$

   gelöst, denn

   $$x_1 \cdot 3 + y_1 \cdot 26 = 3 \cdot 9 - 26 = 1$$

2. $\alpha_2 = 5$. Es gilt:

   $$\text{ggT}(5, 26) = 1$$

   Daher gibt es Zahlen $x_2, y_2 \in \mathbb{Z}$ mit

   $$x_2 \cdot 5 + y_2 \cdot 26 = 1$$

   Diese Gleichung wird durch

   $$x_2 = -5, y_2 = +1$$

   gelöst, denn

   $$x_2 \cdot 5 + y_2 \cdot 26 = -5 \cdot 5 + 26 = 1$$

Der obige konstruktive Beweis liefert die Gleichungen, um aus $x_1, x_2, y; 1$ und y_2 die Zahlen x_3, y_3 abzuleiten. Wir hatten

$$\begin{aligned} x_3 &= x_1 \cdot x_2 = -45 \\ y_3 &= \alpha_1 x_1 y_2 + \alpha_2 x_2 y_1 + 26 y_1 y_2 \\ &= 3 \cdot 9 \cdot 1 + 5 \cdot (-5) \cdot (-1) + 26 \cdot (-1) \cdot 1 \\ &= 27 + 25 - 26 = 26 \end{aligned}$$

Dann ist

$$x_3 \cdot \alpha_3 + 26 \cdot y_3 = -45 \cdot 15 + (26)^2 = -675 + 676 = 1$$

Folgerungen:

① Aus obigen Resultaten folgt, daß die Menge der affinen Chiffren unter der Hintereinanderausführung abgeschlossen ist. Das bedeutet, die Hintereinanderausführung zweier affiner Verschlüsselungen ist äquivalent zu einer einzelnen affinen Verschlüsselung. Wie im Fall der Shift Chiffre bietet daher die zweimalige Verschlüsselung (mit zwei verschiedenen Schlüsseln $(\alpha_1, \beta_1), (\alpha_2, \beta_2)$) eines Klartexts mit der affinen Chiffre keine höhere Sicherheit (zum Beispiel gegenüber Brute-Force Attacken) als eine einfache Verschlüsselung. Der Grund ist der gleiche wie bei der CÄSAR Chiffre. Die zugrundeliegende Gruppenstruktur impliziert, daß die Hintereinanderausführung zweier affiner Chiffren mit Parametern (α_1, β_1) und (α_2, β_2) äquivalent ist zu einer einzigen affinen Verschlüsselung mit Parameter (Schlüssel) $(\alpha_1 \cdot \alpha_2, \alpha_2\beta_1 + \beta_2)$.

② Die in diesem Beispiel durchgeführte Analyse gilt ganz allgemein und zeigt, wie Resultate aus der Untersuchung der algebraischen Struktur einer Verschlüsselung verwendet werden, um Aussagen über die Chiffre zu treffen. Wenn also die Hintereinanderausführung zweier Verschlüsselungen einer Chiffre eine Gruppenstruktur aufweist, bietet die zweifache Verschlüsselung keine höhere Sicherheit als eine einfache Verschlüsselung.

Behauptung:

Die Menge der affinen Chiffren bildet unter der Operation *Hintereinanderausführung* eine nichtkommutative Gruppe.

Beweis:

Gemäß Definition einer Gruppe sind folgende Eigenschaften nachzuweisen:

❶ Abgeschlossenheit

❷ Assoziativität

❸ Neutrales Element

❹ Inverses Element

❺ Nicht-Kommutativität

Anmerkung:

Aus algebraischer Sicht bilden Gruppenstrukturen die Grundlage von Public Key Kryptosystemen. Daher ist es unumgänglich, die Eigenschaften von Gruppen näher zu untersuchen. Diese sehen wir uns in Kapitel [16] näher an.

13.3 Ringe

Wir haben im letzten Abschnitt gesehen, dass die Untersuchung von Mengen und darauf definierten binären Operation zu algebraischen Strukturen führt wie Halbgruppen oder Gruppen. Auf vielen Mengen — wie zum Beispiel $\mathbb{Z}$ — sind jedoch mehr als eine binäre Operation definiert, ganze Zahlen können beispielsweise addiert und multipliziert werden. Diese Eigenschaft führt auf weitere, reichhaltigere algebraische Strukturen.

Definition [13.14]:

Ein **Ring** ist ein Tripel $(\mathbb{A}, +, \times)$ — also eine Menge $\mathbb{A}$ zusammen mit zwei binären Operationen $+$ and $\times$ — das die folgenden Eigenschaften erfüllt:

1. $(\mathbb{A}, +)$ ist eine Abelsche Gruppe.
2. Die Operation $\times$ ist assoziativ, *i.e.*

$$(a \times b) \times c = a \times (b \times c) \quad \forall\, a, b, c \in \mathbb{A}.$$

3. Die *Distributivgesetze* sind erfüllt:

$$a \times (b + c) = a \times b + a \times c \quad \forall\, a, b, c \in \mathbb{A},$$
$$(a + b) \cdot c = a \times c + b \times c \quad \forall\, a, b, c \in \mathbb{A}.$$

Beispiel [13.19]

- ☞ Die Menge der ganzen Zahlen mit Addition und Multiplikation bilden einen Ring.
- ☞ Die Menge der Polynome in x mit Koeffizienten in einem Körper (siehe unten) bilden einen Ring.

Wie oben aufgeführt, ist das offensichtlichste Beispiel eines Rings der Ring der ganzen Zahlen $(\mathbb{Z}, +, \times)$. Betrachtet man die Eigenschaften dieses Rings etwas genauer, stellt man fest, dass $(\mathbb{Z}, +, \times)$ weitere Eigenschaften hat, die ein Ring im allgemeinen nicht hat. Beispielsweise hat der Ring $(\mathbb{Z}, +, \times)$ ein neutrales Element bezüglich der Multiplikation. Dies ist in der obigen Definition nicht gefordert. Daher klassifiziert man Ringe weitergehend wie folgt:

Definition [13.15]:

(a) Ein Ring heißt **Ring mit Identität**, wenn der Ring ein neutrales Element der Multiplikation hat, *i.e.* es gibt ein Element $\mathbf{1}_{\mathbb{A}}$ mit

$$a\mathbf{1}_{\mathbb{A}} = \mathbf{1}_{\mathbb{A}}a = a \qquad \forall\, a \in \mathbb{A}.$$

(b) Ein Ring heißt **kommutativ**, wenn die Operation $\times$ kommutativ ist.

(c) Ein Ring heißt **Integritätsbereich**, wenn der Ring kommutativ ist mit multiplikativem neutralen Element $\mathbf{1}_{\mathbb{A}} \neq 0$. In diesem Ring folgt:

$$\text{Wenn } ab = 0 \implies a = 0 \text{ oder } b = 0.$$

(d) Ein Ring heißt **Schiefkörper** oder **Divisionsring**, wenn die nichtverschwindenden Elemente von $\mathbb{A}$ eine Gruppe unter der Operation $\times$ bilden.

(e) Ein kommutativer Divisionsring nennt man **Körper**.

Anmerkungen:

1. Die Eigenschaft (3) – nämlich dass aus $ab = 0$ folgt: $a = 0$ oder $b = 0$ – bedeutet, dass ein Integritätsbereich **nullteilerfrei** ist. Anders ausgedrückt, in einem Integritätsbereich gibt es keine zwei Elemente $a \neq 0$ und $b \neq 0$, so dass $a \times b = 0$.
2. Man beachte den Unterschied zwischen einem Körper und eine Schiefkörper aus den beiden Definitionen (4) und (5). In einem Schiefkörper ist die Multiplikation nicht kommutativ, was jedoch für einen Körper gelten muß.

Beispiel [13.20]

① Sei R eine Abelsche Gruppe mit Gruppenoperation $+$. Definiere $a \times b = 0$ für alle $a, b \in R$. Dann ist $(R, +, \times)$ ein Ring.

② Die ganzen Zahlen bilden unter der üblichen Addition und Multiplikation einen Integralbereich, aber keinen Körper.

③ Die geraden ganzen Zahlen bilden einen kommutativen Ring ohne Eins.

④ Die Menge der Funktionen von den reellen Zahlen in reelle Zahlen bilden einen kommutativen Ring mit Einselement mit:

$$(f+g)(x) = f(x) + g(x) \qquad (fg)(x) = f(x) \times g(x) \qquad \forall x \in \mathbb{R}.$$

⑤ Die Menge aller 2×2-Matrizen mit reellen Zahlen als Einträge bilden einen nichtkommutativen Ring mit Einselement bezüglich der Matrizenmultiplikation und -addition.

13.4 Körper

Die vertrauteste algebraische Struktur ist die eines Körpers (engl.: *field*).

Definition [13.16]:

Ein **Körper** ist eine nichtleere Menge $\mathbb{A}$ mit zwei binären Operationen auf $\mathbb{A}$, genannt *Addition* $(+)$ und *Multiplikation* $(\cdot)$, mit folgenden Eigenschaften:

1. Es gelten die beiden *Assoziativgesetze*
   Für alle $a, b, c \in \mathbb{A}$:
   $$a + (b + c) = (a + b) + c$$
   $$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$

2. Es gelten die *Kommutativgesetze*
   Für alle $a, b \in \mathbb{A}$:
   $$a + b = b + a$$
   $$a \cdot b = b \cdot a$$

3. Es gelten die beiden *Distributivgesetze*
   Für alle $a, b, c \in \mathbb{A}$:
   $$a \cdot (b + c) = a \cdot b + a \cdot c (a + b) \cdot c \qquad = a \cdot c + b \cdot c$$

4. *Neutrale Elemente*
   Es existieren zwei unterschiedliche spezielle Elemente in $\mathbb{A}$. die **neutralen Elemente**. Eines dieser Elemente heißt **Null Element** oder *neutrales Element der Addition* und wird mit 0_A bezeichnet. Das andere Element heißt **Eins Element** oder *neutrales Element der Multiplikation* und wird mit 1_A bezeichnet. Diese Elemente erfüllen folgende Eigenschaften:

   Für alle $a \in \mathbb{A}$:
   $$a + 0_A = 0_A + a = a$$
   $$a \cdot 1_A = 1_A \cdot a = a$$

5. *Inverse Elemente*
   Für jedes $a \in \mathbb{A}$ gibt es ein Element in $\mathbb{A}$, das mit $-a$ bezeichnet wird und **additives Inverses** a heißt (oder Negatives von a) mit:
   $$a + (-a) = (-a) + a = 0_A$$

   Für jedes $a \neq 0$ $a \in \mathbb{A}$ gibt es ein Element in $\mathbb{A}$, das mit a^{-1} bezeichnet wird und **multiplikatives Inverses** von a heißt, mit der Eigenschaft
   $$a \cdot (a^{-1}) = (a^{-1}) \cdot a = 1_A$$

Lemma:

Jeder Körper $\mathcal{F}$ hat die folgenden Eigenschaften:

1. $a \cdot 0 = 0 \qquad \forall a \in \mathcal{F}$
2. Wenn $a \cdot b = 0$, dann folgt $a = 0$ oder $b = 0$. Das Produkt zweier nichtverschwindeter Elemente eines Körpers ist nicht 0.

Beweis:

1. Es gilt:

$$a \cdot 0 = a \cdot (0 + 0) = a \cdot 0 + a \cdot 0$$

Addiert man das additive Inverse zu $a \cdot 0$ auf beiden Seiten, dann ergibt sich:

$$0 = a \cdot 0 + (-a \cdot 0) = a \cdot 0 + a \cdot 0 + (-a \cdot 0) = a \cdot 0 + 0 = a \cdot 0$$

Daher gilt: $a \cdot 0 = 0$.

2. Angenommen wir haben $a \cdot b = 0$. Falls $a \neq 0$ dann hat a ein multiplikatives Inverses. Dann ist

$$\begin{aligned} b &= 1 \cdot b \\ &= (a^{-1} \cdot a) \cdot b \\ &= a^{-1} \cdot (a \cdot b) \\ &= a^{-1} \cdot 0 \\ &= 0 \end{aligned}$$

Daher: Falls $a \cdot b = 0$ und $a \neq 0$ gilt $b = 0$. Falls $a \cdot b = 0$ und $b \neq 0$ finden wir durch eine analoge Argumentation, daß gelten muß $a = 0$. Damit: $a \cdot b = 0 \implies a = 0$ oder $b = 0$.

Beispiel [13.21]

Die bekanntesten Körper bilden die Mengen $\mathbb{Q} \setminus \{0\}$ der rationalen Zahlen (ohne die 0) mit Addition und Multiplikation, die reellen Zahlen $\mathbb{R}$ und die komplexen Zahlen $\mathbb{C}$.

Im Laufe der folgenden Betrachtungen werden wir weitere Mengen kennenlernen, die mit entsprechenden binären Operationen ebenfalls die algebraische Struktur von Körpern haben. Da diese Mengen endlich sind, spricht man auch von **endlichen Körpern**.[6]

[6] Engl.: *finite fields*.

Kapitel 14

Elementare Eigenschaften von Zahlen

Sehr viele Konzepte aus der Zahlentheorie haben direkte Anwendungen in dem Design kryptographischer Algorithmen. Die Konzepte und Techniken der Zahlentheorie sind ziemlich abstrakt und es ist häufig sehr schwierig, die Resultate intuitiv ohne Beispiele zu verstehen. Daher werden wir die grundlegenden Konzepte, die wir in der Kryptologie benötigen, anhand von Beispielen illustrieren.

Zentrale Konzepte der elementaren Zahlentheorie sind **Teiler**, **Primzahlen**,[1] der größte gemeinsame Teiler zweier Zahlen und **teilerfremde Zahlen**.

14.1 Teiler

Definition [14.17]:

Eine Zahl $b \neq 0$ heisst **Teiler** von $a \in \mathbb{Z}$ und $a \in \mathbb{Z}$ heisst **Vielfaches** von b, falls

$$a = m \cdot b$$

für ein m, wobei a, b und m ganze Zahlen aus $\mathbb{Z}$ sind.

Diese Definition bedeutet also, dass die Zahl b die Zahl a *ohne Rest* teilt. Um diese Tatsache auszudrücken, führt man eine spezielle Notation ein. Man schreibt $b \mid a$, falls a durch die Zahl b teilbar ist. Gleichbedeutend ist auch die Aussage: $b \mid a$ bedeutet, dass b die Zahl a teilt. Häufig betrachtet man die Menge der Teiler einer gegebenen Zahl $a \in \mathbb{Z}$.

[1] Siehe auch die URL:

`http://www.utm.edu/research/primes`

für weitergehende Informationen über Primzahlen.

Beispiel [14.22]

Die positiven Teiler von $a = 24$ sind die Zahlen 1, 2, 3, 4, 6, 8, 12, und 24.

Es gelten die folgenden Aussagen, die direkte Konsequenzen der Definition des Teilers sind:

- Wenn $a \mid 1$ dann $a = \pm 1$
- Wenn $a \mid b$ und $b \mid a$ dann folgt $a = \pm b$
- Jedes $b \neq 0$ ist Teiler von 0.
- Wenn $b \mid g$ und $b \mid h$ dann folgt $b \mid (mg + nh)$

Um die letzte Aussage zu zeigen, beachten wir, dass gilt:

Wenn $b \mid g$ dann ist gemäß Definition $g = b \cdot g_1$ für ein geeignetes $g_1 \in \mathbb{Z}$.

Wenn $b \mid h$ dann ist gemäß Definition $h = b \cdot h_1$ für ein geeignetes $h_1 \in \mathbb{Z}$.

Daher ist

$$m \cdot g + n \cdot h = m \cdot b \cdot g_1 + n \cdot b \cdot h_1 = b \cdot (m \cdot g_1 + n \cdot h_1)$$

und b ist Teiler von $mg + nh$.

Anmerkungen:

Man nennt Zahlen *vollkommene Zahlen*,[2] wenn eine Zahl als Summe ihrer echten Teiler geschrieben werden kann. Echte Teiler sind alle Teiler ohne die Zahl selbst.[3]

Beispielsweise ist die Zahl 6 eine vollkommene Zahl, denn die echten Teiler von 6 sind 1, 2 und 3 und $6 = 1 + 2 + 3$. Eine weitere vollkommene Zahl ist 28, denn

$$28 = 1 + 2 + 4 + 7 + 14$$

Die ersten sechs vollkommene Zahlen sind:

$$\begin{aligned} 6 &= 1 + 2 + 3 \\ 28 &= 1 + 2 + 4 + 7 + 14 \\ 496 &= 1 + 2 + 4 + 8 + \ldots + 248 \\ 8\,128 &= 1 + 2 + 4 + 8 + \ldots + 4064 \\ 33\,550\,336 &= 1 + 2 + 4 + 8 + \ldots + 16\,775\,168 \\ 8\,589\,869\,056 &= 1 + 2 + 4 + 8 + \ldots \end{aligned}$$

Bereits die Griechen[4] haben ein Verfahren gekannt, wie man vollkommene Zahlen konstruieren kann:

[2] Engl.: *perfect numbers*.
[3] Siehe z.B. [232, pp. 160-161] oder [195, p. 34].
[4] Dies findet man beispielsweise in EUKLIDs *Elemente*, Buch IX, Behauptung 36.

1. Beginne mit der Zahl 1 und addiere solange aufeinanderfolgende Potenzen von 2 bis man eine Primzahl erhält.

$$1 + 2 + 4 + 8 + \ldots + 2^n$$

2. Dann ist die Zahl

$$N = 2^n\Big(1 + 2 + 4 + 8 + \ldots + 2^n\Big)$$

eine vollkommene Zahl

So haben wir beispielsweise

$$1 + 2 + 4 = 2^0 + 2^1 + 2^2 = 7 \qquad \text{Primzahl}$$

und

$$N = 2^2 \cdot (1 + 2 + 4) = 28$$

ist wie wir bereits wissen, eine vollkommene Zahl. Oder

$$1 + 2 + 4 + 8 + 16 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 31 \qquad \text{Primzahl}$$

womit folgt:

$$N = 2^4 \cdot \Big(1 + 2 + 4 + 8 + 16\Big) = 496$$

ist ebenfalls eine vollkommene Zahl.

EUKLID hat in den *Elementen* gezeigt — *i.e.* bewiesen — dass eine derart konstruierte Zahl eine vollkommene Zahl ist. Er hat aber nicht behauptet, dass dies die *einzigen* perfekten Zahlen sind. Das ist natürlich eine Herausforderung an die Mathematiker, erst im 18. Jahrhundert gelang es LEONHARD EULER zu zeigen, dass alle geraden, vollkommenen Zahlen die von EUKLID gezeigte Form haben. Bis heute ist offen und ungeklärt, ob es ungerade vollkommene Zahlen gibt.[5]

14.1.1 Übungen

Übung 14.1:

Seien $m, n \in \mathbb{Z}$ und $k \in \mathbb{N}$. Sind die folgenden Aussagen wahr oder falsch? Begründen Sie Ihre Antwort.

(a) Wenn k ein Teiler von $m + n$ ist, dann auch jeweils von n und m.

(b) Wenn k ein Teiler von $m \cdot n$ ist, dann auch jeweils von n und m.

(c) Wenn k ein Teiler von $m \cdot n$ ist, dann auch von n oder von m.

[5] Siehe [52], Theorem 1.3.3, für einen Beweis dieses Satzes von EUKLID und EULER in moderner Notation.

(d) Wenn k zwar m teilt aber nicht n, dann ist k auch kein Teiler von $n+m$.

(e) Wenn k zwar m teilt aber nicht n, dann ist k auch kein Teiler von $n \cdot m$.

(f) Wenn k jeweils n und m mit Rest 1 teilt, dann teilt k auch $n \cdot m$ mit Rest 1.

(g) Wenn k jeweils n und m mit Rest 1 teilt, dann teilt k auch $n+m$ mit Rest 1.

14.2 Primzahlen

Eine ganze Zahl $p \in \mathbb{Z}, p > 1$ ist eine **Primzahl**, genau dann, wenn die einzigen Teiler von p die Zahlen ± 1 und $\pm p$ sind. Die Zahl 1 zählt nicht zu den Primzahlen. Eine Zahl $n > 1$, welche keine Primzahl ist, nennt man **zusammengesetzte Zahl**. Die Primzahlen spielen eine herausragende Rolle sowohl in der Zahlentheorie als auch in der Kryptologie. In der Tabelle [14.1] sind die 109 Primzahlen unter 600 aufgelistet.

2	101	211	307	401	503
3	103	223	311	409	509
5	107	227	313	419	521
7	109	229	317	421	523
11	113	233	331	431	541
13	127	239	337	433	547
17	131	241	347	439	557
19	137	251	349	443	563
23	139	257	353	449	569
29	149	263	359	457	571
31	151	269	367	461	577
37	157	271	373	463	587
41	163	277	379	467	593
43	167	281	383	479	599
47	173	283	389	487	
53	179	293	397	491	
59	181			499	
61	191				
67	193				
71	197				
73	199				
79					
83					
89					
97					

Tabelle 14.1: Die Primzahlen unter 600.

Primzahlen haben aus drei Gründen besondere Bedeutung:

① Primzahlen werden in der mathematischen Disziplin der Zahlentheorie als die Grundbausteine der natürlichen Zahlen betrachtet. Primzahlen bilden die Grundlage vieler mächtiger Sätze der Zahlentheorie.

② Primzahlen haben in der modernen Kryptographie — insbesondere der Public–Key Kryptographie — eine große praktische Bedeutung erlangt. Das meist verbreitete Public–Key Kryptosystem ist die RSA Verschlüsselung, die Ende der siebziger Jahre entwickelt wurde. Die Sicherheit dieses Verfahrens hängt entscheidend von der Wahl von (großen) Primzahlen ab, die für bestimmte Parameter benutzt werden. Weitere, anschließend entwickelte Verfahren für Verschlüsselung oder digitale Signatur benutzen ebenfalls Primzahlen als Grundlage des Krypto–Algorithmus.

③ Keine unmittelbare praktische Bedeutung hat die computergestützte Suche nach der größten Primzahl. Dennoch liefert diese Suche eine Reihe von Benchmarktests für Supercomputer oder Verfahren zur Parallelisierung der Berechnung auf verteilten Systemen.

Die Mathematiker beschäftigen sich bereits sehr lange mit Primzahlen. Ein wichtiges Resultat geht zurück auf EUKLID, der vor mehr als 2000 Jahren bereits gezeigt hat, dass es unendlich viele Primzahlen gibt.

Theorem [3]:

Es gibt unendlich viele Primzahlen.

Beweis:

Der Beweis dieses Satzes von EUKLID ist *der* klassische Widerspruchsbeweis.

- Annahme: Es gibt nur endlich viele Primzahlen.
- Wenn es endlich viele Primzahlen gibt, dann können diese aufgezählt und nach Größe sortiert werden, *i.e.*:

$$\mathbb{P} = \{2, 3, 5, 7, 11, 13, \ldots, p_n\}.$$

- Bilde nun die Zahl:

$$\begin{aligned} q &= 2 \cdot 3 \cdot \ldots \cdot p_n + 1 \\ &= \prod_{i=1}^{n} p_i + 1. \end{aligned}$$

- Da die Menge $\mathbb{P}$ alle Primzahlen enthält, hat die Zahl q Teiler aus der Menge $\mathbb{P}$.
- Durch welche Primzahl aus der Menge $\mathbb{P}$ man die Zahl q aber teilt, es bleibt immer der Rest 1.

- Keine Primzahl aus der Menge $\mathbb{P}$ kann Teiler von q sein, q hat also einen anderen Primteiler oder ist selbst eine Primzahl.

- Der letzte Schritt ist ein Widerspruch zur Annahme, dass die Menge $\mathbb{P}$ alle Primzahlen enthält. Die Annahme, es gibt nur endlich viele Primzahlen ist daher falsch.

- Daraus folgt, es gibt unendlich viele Primzahlen.

Beispiel [14.23]

Um die Logik des Beweises von EUKLID besser nachvollziehen zu können, nehmen wir an, die Menge der Primzahlen ist

$$\mathbb{P} = \{3, 5, 7\} = \{p_1, p_2, p_3\}$$

Die im Beweis benutzte Konstruktion führt auf die Zahl

$$q = p_1 p_2 p_3 + 1 = 2 \cdot 5 \cdot 7 + 1 = 106.$$

Man prüft nun leicht nach:

① Wird 106 durch $p_1 = 3$ geteilt, bleibt Rest 1

② Wird 106 durch $p_2 = 5$ geteilt, bleibt Rest 1

③ Wird 106 durch $p_3 = 7$ geteilt, bleibt Rest 1

Daher hat

$$106 = 2 \cdot 53$$

andere Primteiler, die nicht in der Menge $\mathbb{P}$ enthalten sind.

Eine Zahl, die keine Primzahl ist, nennt man eine **zusammengesetzte Zahl**.

Die zentrale Rolle, die Primzahlen in der Zahlentheorie spielen, hängt letztendlich damit zusammen, dass die Primzahlen die 'Atome' der Zahlen darstellen. Ein zentraler Satz der Zahlentheorie — der **Fundamentalsatz der Zahlentheorie** — sagt aus, dass jede natürliche Zahl $a \in \mathbb{N}$ (größer 2) eindeutig (bis auf die Reihenfolge der Faktoren) als Produkt von Potenzen von Primzahlen geschrieben werden kann.

Theorem [4]:

Fundamentalsatz der Zahlentheorie

Jede natürliche Zahl $a > 2$ lässt sich als Produkt von Primzahlen darstellen.

$$a = \prod_{i=1}^{n} p_i^{\alpha_i} \qquad = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_n^{\alpha_n}$$

mit den Primzahlen $p_1 < p_2 < \ldots < p_n$ und jede Potenz $\alpha_i > 0$.

Die Zerlegung einer Zahl $a \in \mathbb{N}$ in ihre Primfaktoren nennt man **Faktorisierung**.

In Kapitel [19] werden wir weitere Eigenschaften von Primzahlen untersuchen.

Anmerkung:

Die im letzten Abschnitt betrachteten vollkommenen Zahlen werden konstruiert, indem die Summe von 2er Potenzen betrachtet werden, die sich zu einer Primzahl aufaddieren. Solche Primzahlen nennt man **Mersenne Primzahlen**, benannt nach dem französischen Geistlichen Martin Mersenne (1588 – 1648), der diese speziellen Primzahlen 1644 diskutierte. Eine umfassende Diskussion dieser Primzahlen findet man in [52, Chap. 1.3.1].

14.3 Der größte gemeinsame Teiler zweier Zahlen

Definition [14.18]:

(a) Seien $a, b \in \mathbb{Z}$ und sei $k \in \mathbb{Z}$ so, dass k sowohl die Zahl a als auch die Zahl b teilt. Dann ist k ein **gemeinsamer Teiler** von a und b. Analog heißt eine Zahl $v \in \mathbb{Z}$, die sowohl von a und von b geteilt wird, ein **gemeinsames Vielfaches** von a und b.

(b) Sei $a \neq 0$ oder $b \neq 0$. Dann bezeichnen wir die größte Zahl $k \in \mathbb{N}$, die gemeinsamer Teiler von a und b ist, als den **größten gemeinsamen Teiler** von a und b und schreiben für diesen Scahverhalt

$$k = \mathtt{ggT}(a, b). \tag{14.1}$$

(c) Sei $a \neq 0$ und $b \neq 0$. Die kleinste natürliche Zahl v, die ein gemeinsames Vielfaches von a und b ist, heißt das **kleinste gemeinsame Vielfache** von a und b und schreiben

$$v = \mathtt{kgV}(a, b). \tag{14.2}$$

(d) Wir definieren

$$\begin{aligned} \texttt{ggT}(0,0) &= 0, \\ \texttt{kgV}(a,0) &= \texttt{kgV}(0,a) = 0 \end{aligned}$$

für alle $a \in \mathbb{Z}$.

Eine positive ganze Zahl k heißt **größter gemeinsamer Teiler** von a und b, *i.e.* $k = \texttt{ggT}(a,b)$, wenn gilt:

① k ist ein Teiler von a und b.

② Jeder andere Teiler von a und b ist auch Teiler von k.

Eine äquivalente Formulierung ist:

$$\boxed{\texttt{ggT}(a,b) = \max_k\{k \in \mathbb{Z} \mid \text{ so dass } k \mid a \text{ und } k \mid b\}}$$

Da der größte gemeinsame Teiler per Definition positiv ist, folgt:

$$\texttt{ggT}(a,b) = \texttt{ggT}(a,-b) = \texttt{ggT}(-a,b) = \texttt{ggT}(-a,-b).$$

Beispiel [14.24]

$$\texttt{ggT}(60,24) = \texttt{ggT}(60,-24) = 12$$

Es gibt mehrere Verfahren um den größten gemeinsamen Teiler zweier ganzen Zahlen zu finden. Die Schulmethode besteht darin, die ganzen Zahlen in das Produkt ihrer Primfaktoren zu zerlegen.

Beispiel [14.25]

Wenn man den größten gemeinsamen Teiler von 36 und 300 finden will, führt man folgende Zerlegung durch:

$$\begin{aligned} 300 &= 2^2 \cdot 3^1 \cdot 5^2 \\ 36 &= 2^2 \cdot 3^2 \\ \texttt{ggT}(36,300) &= 2^2 \cdot 3^1 \cdot 5^0 = 12 \end{aligned}$$

Generell gilt daher:

$$k = \texttt{ggT}(a,b) \longrightarrow k_p = \min(a_p, b_p) \qquad \forall\, p \in P$$

Will man dieses Verfahren zur Bestimmung des größten gemeinsamen Teilers auf große Zahlen anwenden, führt dies auf das Problem langer Laufzeiten, da die bekannten Faktorisierungsmethoden — *i.e.* die Verfahren zur Zerlegung zusammengesetzte Zahlen in ihre Primfaktoren — sämtlich die Eigenschaft haben, dass sie nichtpolynomiales Laufzeitverhalten haben. Eine alternative Methode zur Bestimmung des ggTs zweier Zahlen — die wir in Abschnitt [18] untersuchen werden — ist der EUKLIDische Algorithmus.

Theorem [5]:

Lemma von Bézout

Seien a und b ganze Zahlen. Dann gibt es Zahlen $x, y \in \mathbb{Z}$ mit:

$$\text{ggT}(a,b) = x \cdot a + y \cdot b. \tag{14.3}$$

Beweis:

Seien a und b gegeben und sei d die kleinste positive Zahl der Form

$$d = a \cdot x + b \cdot y$$

mit $x, y \in \mathbb{Z}$. Ein solches d existiert immer, z.B. $d = a^2 + b^2$.

Behauptung: $d = \text{ggT}(a,b)$

Falls d' irgend ein Teiler von a und b ist, dann ist d' auch Teiler der Linearkombination

$$d' \mid ax + by$$

denn, wenn $d' \mid a$, dann folgt $a = k_1 \cdot d'$. Ist $d' \mid b$ dann ist per Definition $b = k_2 \cdot d'$. Dann ist aber

$$\begin{aligned} ax + by &= ak_1 d' + bk_2 d' \\ &= (a \cdot k_1 + b \cdot k_2) \cdot d' \end{aligned}$$

oder

$$d' \mid (ax + by)$$

14.4 Relativ prime Zahlen

Eine wichtige Rolle in der Zahlentheorie spielen Zahlen, die außer der 1 keinen gemeinsamen Teiler haben.

Definition [14.19]:

Zwei Zahlen $a, b \in \mathbb{Z}$ heißen **relativ prim** oder **coprim** oder **relativ zueinander prim** oder **teilerfremd**, falls sie keinen Primfaktor gemeinsam haben.

Bemerkung:

Zwei Zahlen sind coprim, falls ihr einziger gemeinsamer Faktor die 1 ist. Dies ist äquivalent zu der Aussage: a und b sind coprim, falls der größte gemeinsame Teiler dieser Zahlen 1 ist, *i.e.*

$$\text{ggT}(a, b) = 1$$

Beispiel [14.26]

Die Zahlen 8 und 15 sind coprim, denn die Teiler von 8 sind 1, 2, 4 und 8 und die Teiler von 15 sind 1, 3, 5, 15: daher ist 1 die einzige Zahl, die 8 und 15 teilt.

14.5 Die Eulersche Totientenfunktion

Ein wichtiges Konzept in der Zahlentheorie — und in der Public–Key Kryptographie — ist die sogenannte **Euler Funktion**, die man auch EULERsche ϕ–Funktion oder EULERsche Totienten Funktion nennt. Diese Funktion wurde von dem Schweizer Mathematiker LEONHARD EULER (1707 – 1783) eingeführt.

Die Totientenfunktion hängt von einer Zahl n ab und gibt die Anzahl der positiven Zahlen an, kleiner als n und relativ prim zu n. Um ein Gefühl dafür zu erhalten, wie man mit der ϕ–Funktion arbeitet, sehen wir uns zunächst einige Beispiele an:

❶ $n = 5$

$$\phi(5) = 4$$

Da die vier Zahlen 1, 2, 3 und 4 coprim zu 5 sind, gibt es also 4 Zahlen kleiner 5 mit dieser Eigenschaft.

❷ $n = 18$ Die Totientenfunktion von $n = 18$ ist

$$\phi(18) = 6,$$

da die sechs Zahlen 1, 5, 7, 11, 13 und 17 teilerfremd zu 18 sind.

❸ $n = 10$

$$\phi(10) = 4,$$

denn die vier Zahlen 1, 3, 7, und 9 sind coprim zu 10.

Die Tabelle [14.2] gibt eine Liste der ersten 30 Werte von $\phi(n)$ an. Der Wert $\phi(1) = 1$ ist dabei ohne Bedeutung.

n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$
1	1	11	10	21	12
2	1	12	4	22	10
3	2	13	12	23	22
4	2	14	6	24	8
5	4	15	8	25	20
6	2	16	8	26	12
7	6	17	16	27	18
8	4	18	6	28	12
9	6	19	18	29	28
10	4	20	8	30	8

Tabelle 14.2: Die ersten Werte der EULERschen Totienten Funktion $\phi(n)$.

Es ist offensichtlich — da keine Zahl kleiner als diese Primzahl einen Teiler $\neq 1$ mit der Primzahl gemeinsam hat — dass für eine Primzahl p immer gilt:

$$\phi(p) = p - 1.$$

Aus der Tabelle [14.2] liest man unmittelbar ab, dass für $n > 2$ die Werte der EULERschen Totientenfunktion gerade sind. Dies ist immer der Fall. Ist a eine Zahl, die in $\phi(n)$ gezählt wird, dann ist $n - a$ auch eine Zahl, die teilerfremd zu n ist und damit von $\phi(n)$ gezählt wird. Das gilt aufgrund der Eigenschaft

$$\mathrm{ggT}(a, n) = \mathrm{ggT}(n - a, n).$$

Außerdem sind die beiden Zahlen a und $n - a$ verschieden. Denn, nehmen wir das Gegenteil an, dann sind also die beiden Zahlen a und $n - a$ gleich, dann ist aber $a = n - a$ oder $n = 2a$. Dies ist aber inkonsistent zur Annahme, dass

$$\mathrm{ggT}(a, n) = 1$$

(denn a wird ja durch $\phi(n)$ gezählt). Daher sind a und $n - a$ zwei verschiedene Zahlen, die durch $\phi(n)$ gezählt werden.

Eine weitere Eigenschaft der EULER Funktion, die aus der Tabelle [14.2] abgelesen werden kann, ist die Tatsache, dass die Totientenfunktion in bestimmten Fällen multiplikativ ist. Beispielsweise liest man ab:

$$\phi(3) \cdot \phi(7) = 2 \cdot 6 = 12 = \phi(21),$$
$$\phi(4) \cdot \phi(5) = 2 \cdot 4 = 8 = \phi(20).$$

Andererseits gibt es Fälle, bei denen dies nicht funktioniert:

$$\phi(4) \cdot \phi(6) \neq \phi(24).$$

Die korrekte Aussage lautet daher:

Theorem [6]:

Wenn $\mathrm{ggT}(m,n) = 1$ dann ist $\phi(m) \cdot \phi(n) = \phi(m \cdot n)$.

Einen rigorosen Beweis dieses Satzes findet man zum Beispiel in [142, pp. 44]. Wir betrachten einen Spezialfall dieses Satzes, der für die Public–Key Kryptographie wichtig ist.

Corollar:

Wenn p und q zwei Primzahlen sind und $n = p \cdot q$ dann gilt:

$$\phi(n) = \phi(p \cdot q) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1).$$

Beweis:

Sind p und q Primzahlen und $n = p \cdot q$, dann ist die Menge $\mathbb{Z}_n$ gegeben durch die $p \times q$ Zahlen

$$\mathbb{Z}_n = \{0, 1, 2, \ldots, (p \cdot q - 1)\}.$$

Von diesen Zahlen sind die folgenden Zahlen nicht coprim zu n:

$$\{p, 2p, \ldots, (q-1)p)\},$$

die Menge

$$\{q, 2q, \ldots, (p-1)q)\}$$

und 0. Daher

$$\begin{aligned}\phi(n) &= p \cdot q - [(q-1) + (p-1) + 1] \\ &= p \cdot q - (p+q) + 1 \\ &= (p-1) \cdot (q-1) \\ &= \phi(p) \cdot \phi(q)\end{aligned}$$

Theorem

Ist n eine natürliche Zahl und

$$n = \prod_{j=1}^{k} p_j^{\alpha_j} = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$$

die Primfaktorzerlegung von n, dann ist

$$\phi(n) = \prod_{j=1}^{k} (p_j - 1) \cdot p_j^{\alpha_j - 1}$$

Beweis:

Beispiele:

❶ Betrachte $n = 30$. Da 30 die Zerlegung

$$30 = 2 \cdot 3 \cdot 5 = p_1^1 \cdot p_2^1 \cdot p_3^1$$

hat, erhalten wir:

$$\begin{aligned}
\phi(30) &= \prod_{j=1}^{k} (p_j - 1) \cdot p_j^{\alpha_j - 1} \\
&= (p_1 - 1) \cdot p_1^{\alpha_1 - 1} \cdot (p_2 - 1) \cdot p_2^{\alpha_2 - 1} \cdot (p_3 - 1) \cdot p_3^{\alpha_3 - 1} \\
&= (2 - 1) \cdot 2^0 \cdot (3 - 1) \cdot 3^0 \cdot (5 - 1) \cdot 5^0 \\
&= 8
\end{aligned}$$

was mit dem Eintrag in der Tabelle [14.2] übereinstimmt.

Kapitel 15

Kongruenzen

15.1 Kongruenzrelationen

Um das Konzept der **Kongruenzen** einzuführen, betrachten wir die Gruppe $(\mathbb{Z}, +)$. Für ein festes $n \in \mathbb{N}$ definieren wir die folgende Relation auf $\mathbb{Z} \times \mathbb{Z}$:

$$\equiv_n \subseteq \mathbb{Z} \times \mathbb{Z}$$

mit

$$x \equiv_n y \text{ genau dann wenn } n \mid x - y$$

Mit anderen Worten, zwei ganze Zahlen $x, y \in \mathbb{Z}$ stehen in Relation, genau dann, wenn x und y durch die gleiche Zahl n geteilt werden, bleibt der gleiche Rest. Es gibt also zwei ganze Zahlen $q_x, q_y \in \mathbb{Z}$ und eine Zahl $r \in \mathbb{N}_0$ (der Rest) so daß

$$x = n \cdot q_x + r$$
$$y = n \cdot q_y + r.$$

Die beiden ganzen Zahlen x und y heißen *kongruent modulo* n. Für diese Tatsache führt man die Schreibweise ein:

$$x \equiv y \bmod n.$$

Theorem [7]:

Die Kongruenz modulo einer festen Zahl $n \in \mathbb{N}$ ist eine *Äquivalenzrelation* auf dem kartesischen Produkt $\mathbb{Z} \times \mathbb{Z}$.

Beweis:

Um zu zeigen, dass die Kongruenzrelation modulo n eine Äquivalenzrelation auf $\mathbb{Z} \times \mathbb{Z}$ ist, müssen wir drei Eigenschaften zeigen:

1. Reflexivität
2. Symmetrie
3. Transitivität

1: Reflexivität bedeutet, $a \equiv a \mod n$. Dies ist aber trivial, da

$$n \mid (a - a)$$

2: Symmetrie bedeutet, wenn

$$a \equiv b \bmod n$$

dann folgt

$$b \equiv a \bmod n$$

Die ist ebenfalls leicht zu verifizieren, denn, wenn $a \equiv b \bmod n$, dann ist n Teiler von $a - b$, *i.e.* $n \mid a - b$. Das impliziert aber $n \mid -(b - a)$ und $n \mid b - a$; daher $b \equiv a \bmod n$.

3: Schließlich bedeutet die Transitivität, wenn

$$a \equiv b \mod n$$
$$b \equiv c \mod n$$

gilt, dann müssen wir zeigen, dass auch

$$a \equiv c \mod n$$

gilt.

Wenn also gemäß Voraussetzung gilt

$$a \equiv b \mod n$$
$$b \equiv c \mod n$$

dann folgt, dass n Teiler der Differenzen $a - b$ und $b - c$ ist. Wenn jedoch $n \mid (a - b)$ und $n \mid (b - c)$ dann können wir zwei ganze Zahlen k, l finden, die

$$a - b = k \cdot n$$
$$b - c = l \cdot n$$

erfüllen. Addiert man diese beiden Gleichungen, sieht man, dass b wegfällt, wir erhalten daher

$$a - c = (k + l) \cdot n$$

Dies führt auf:

$$a \equiv c \mod n$$

Corollar:

Seien a und b ganze Zahlen; die folgenden Aussagen sind äquivalent:

1. $a \equiv b \bmod n$
2. $a - b$ ist teilbar durch n ohne Rest
3. Es gibt eine ganze Zahl $k \in \mathbb{Z}$ mit $a = b + k \cdot n$
4. a und b haben den gleichen Rest, wenn sie durch n geteilt werden.

Beispiel [15.27]

1. $5 \equiv 3 \bmod 2$
2. $178 \equiv 17 \bmod 7$
3. $-4 \equiv 12 \bmod 4$
4. $15 \equiv 0 \bmod 5$
5. $4 \not\equiv 6 \bmod 3$
6. $19 \equiv 14 \equiv 9 \equiv 4 \equiv -1 \equiv -6 \bmod 5$

15.2 Restklassen

Wenn man mit der Kongruenzrelation modulo einer Zahl n arbeitet, zerfällt die Menge der ganzen Zahlen $\mathbb{Z}$ in n Mengen, die man **Restklassen**[1] nennt. Elemente aus der gleichen Restklasse sind kongruent zueinander, Elemente aus zwei unterschiedlichen Restklassen sind nicht kongruent. Für viele Eigenschaften ist es völlig unerheblich, welches Element aus einer bestimmten Restklasse verwendet wird. Es reicht in den meisten Fällen aus, irgendeinen Vertreter aus der jeweiligen Restklasse zu betrachten. Untersucht man die *Menge aller Restklassen*, dann betrachtet man also eine Menge mit einem Element aus jeder Klasse. Ein solches System $a_1, a_2, \ldots, a_n$ nennt man **vollständiges Restklassensystem modulo** n. Dieses System ist durch folgende Eigenschaften gekennzeichnet:

1. Falls $i \neq j$, dann $a_i \not\equiv a_j \bmod n$
2. Falls $a \in \mathbb{Z}$, dann gibt es ein Index i mit $1 \leq i \leq n$ mit

$$a \equiv a_i \bmod n$$

Der Prototyp eines vollständigen Restklassensystems modulo n ist die Menge der Zahlen

$$\mathbb{Z}_n \stackrel{\text{def}}{=} \{0, 1, 2, 3, \ldots, n-1\}$$

Die Äquivalenzklassen der Kongruenzrelation werden festgelegt durch die Reste, die auftreten, wenn eine beliebige Zahl $a \in \mathbb{Z}$ durch n geteilt wird. Dies führt zu den folgenden Restklassen (dies sind also Mengen von Zahlen):

$$\begin{aligned}
[0]_{\equiv_n} &= \{x \in \mathbb{Z} \mid x = n \cdot q,\ q \in \mathbb{Z}\} \\
[1]_{\equiv_n} &= \{x \in \mathbb{Z} \mid x = n \cdot q + 1,\ q \in \mathbb{Z}\} \\
[2]_{\equiv_n} &= \{x \in \mathbb{Z} \mid x = n \cdot q + 2,\ q \in \mathbb{Z}\} \\
\vdots \qquad & \qquad \vdots \\
[n-1]_{\equiv_n} &= \{x \in \mathbb{Z} \mid x = n \cdot q + (n-1),\ q \in \mathbb{Z}\}
\end{aligned}$$

Diese Äquivalenzklassen heißen **Restklassen modulo** n. Die Menge dieser Klassen bezeichnet man mit $\mathbb{Z}_n$:

$$\mathbb{Z}_n = \{[0]_{\equiv_n}, [1]_{\equiv_n}, [2]_{\equiv_n} \cdots, [n-1]_{\equiv_n}\}$$

Bemerkung:

In der Literatur ist es üblich — um die Notation handhabbarer zu gestalten — die Schreibweise

$$\mathbb{Z}_n \stackrel{\text{def}}{=} \{0, 1, 2, 3, \ldots, n-1\}$$

für die Menge der Restklassen zu verwenden. Diese Schreibweise haben wir auch oben verwendet.

[1] Engl.: *residue classes*.

Beispiel [15.28]

Sei $n = 5$; wir konstruieren das vollständige Restklassensystem modulo 5.

$$\begin{aligned}
[0] &= \{y \in \mathbb{Z} \mid y \text{ geteilt durch } 5 \text{ mit ganzzahligem Rest } 0\} \\
&= \{\ldots, -15, -10, -5, 0, 5, 10, 15, \ldots\} \\
&= \{x \in \mathbb{Z} \mid x = 5 \cdot q;\ q \in \mathbb{Z}\} \\
[1] &= \{y \in \mathbb{Z} \mid y \text{ geteilt durch } 5 \text{ mit ganzzahligem Rest } 1\} \\
&= \{\ldots, -19, -14, -9, -4, 1, 6, 11, 16, \ldots\} \\
&= \{x \in \mathbb{Z} \mid x = 5 \cdot q + 1;\ q \in \mathbb{Z}\} \\
[2] &= \{y \in \mathbb{Z} \mid y \text{ geteilt durch } 5 \text{ mit ganzzahligem Rest } 2\} \\
&= \{\ldots, -18, -13, -8, -3, 2, 7, 12, 17, \ldots\} \\
&= \{x \in \mathbb{Z} \mid x = 5 \cdot q + 2;\ q \in \mathbb{Z}\} \\
[3] &= \{y \in \mathbb{Z} \mid y \text{ geteilt durch } 5 \text{ mit ganzzahligem Rest } 3\} \\
&= \{\ldots, -17, -12, -7, -2, 3, 8, 13, 18, \ldots\} \\
&= \{x \in \mathbb{Z} \mid x = 5 \cdot q + 3;\ q \in \mathbb{Z}\} \\
[4] &= \{y \in \mathbb{Z} \mid y \text{ geteilt durch } 5 \text{ mit ganzzahligem Rest } 4\} \\
&= \{\ldots, -16, -11, -6, -1, 4, 9, 14, 19, \ldots\} \\
&= \{x \in \mathbb{Z} \mid x = 5 \cdot q + 4;\ q \in \mathbb{Z}\}
\end{aligned}$$

Anmerkungen:

❶ Man beachte, dass $[i], i = 0, 1, 2, 3, 4$ *Mengen* von Zahlen sind.

❷ Vereinigt man diese Mengen, dann gilt:

$$\mathbb{Z} = \bigcup_{i=0}^{4} [i] = [0] \cup [1] \cup [2] \cup [3] \cup [4].$$

❸ Bildet man Schnittmengen, dann folgt, dass die Restklassen stets paarweise disjunkt sind:

$$[i] \cap [j] = \emptyset,\ i \neq j, i, j = 0, 1, 2, 3, 4.$$

Wie oben bereits erwähnt, wenn man mit diesen Restklassen arbeitet — da man hier mit *Zahlenmengen* operiert — kann man mit einem beliebigen Vertreter einer Restklasse arbeiten.

Auf der Menge der Restklassen modulo n, $\mathbb{Z}_n$, lassen sich zwei binäre Operationen einführen. Diese heißen **Addition modulo n** und **Multiplikation modulo n** . Diese beiden Operationen auf $\mathbb{Z}_n$ führen zur sogenannten **Arithmetik modulo n**.

Definition [15.20]:

Die binäre Operation

$$\begin{aligned} \oplus_n : \mathbb{Z}_n \times \mathbb{Z}_n &\longrightarrow \mathbb{Z}_n \\ x, y &\longmapsto x \oplus y = x + y \mod n \end{aligned} \tag{15.1}$$

ist die Addition modulo n, und

$$\begin{aligned} \otimes_n : \mathbb{Z}_n \times \mathbb{Z}_n &\longrightarrow \mathbb{Z}_n \\ x, y &\longmapsto x \otimes y = x \cdot y \mod n \end{aligned} \tag{15.2}$$

ist die Multiplikation modulo n.

Die Menge $\mathbb{Z}_n$ mit den Operationen der Addition und Multiplikation modulo n bezeichnet man als **ganze Zahlen modulo n**. Es ist üblich — wenn der Zusammenhang klar ist — die üblichen Symbole '$+$' und '$\cdot$' anstelle von $\oplus_n$ und $\otimes_n$ zu verwenden.

Wenn n nicht allzu groß ist, kann die Addition und Multiplikation modulo n durch Tabellen angegeben werden Wir betrachten einige Beispiele im Detail.

Beispiel [15.29]

Wir betrachten die Menge

$$\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}.$$

Die Addition modulo 5 ist durch die Tabelle [15.1] gegeben.

$+$	**0**	**1**	**2**	**3**	**4**
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Tabelle 15.1: Addition modulo 5 auf $\mathbb{Z}_5$.

In der Tabelle [15.1] ist der Eintrag in der Zeile, die mit r beginnt und die Spalte mit der Beschriftung s die Summe $r + s$ in dem Sinn, dass dieser Eintrag der Repräsentant derjenigen Klasse modulo 5 darstellt, die diese Summe enthält. Daher ist $3 + 3 = 6 \equiv 1 \,(\mod 5)$. Auf ähnliche Weise erhält man $4 + 4 = 8 \equiv 3 \,(\mod 5)$. Die Einträge dieser Tabelle [15.1] werden folgendermaßen berechnet: Wenn die dritte Spalte mit der **2** mit der fünften Zeile mit der **4** addieren (mod 5), erhält man die 1 in der Tabelle aufgrund der Addition

$$2 + 4 = 6 \equiv 1 \mod 5$$

Auf die gleiche Weise finden wir

$$4 + 4 = 8 \equiv 3 \bmod 5$$

was auf den Eintrag '3' in der letzten Zeile und letzten Spalte führt.

×	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Tabelle 15.2: Multiplikation auf der Menge $\mathbb{Z}_5$.

Führt man ein analoges Verfahren für die Multiplikation durch, erhält man die Tabelle [15.2]. Die Einträge dieser Tabelle werden beispielsweise folgendermaßen berechnet:

$$2 \cdot 4 = 8 \equiv 3 \bmod 5$$

Analog erhält man:

$$4 \cdot 4 = 16 \equiv 1 \bmod 5$$

Dies ist die '1' der letzten Zeile und letzten Spalte der Tabelle [15.2].

Anmerkungen:

- Die modulare Multiplikation ist vielleicht eine ungewohnte Operation, die modulare Addition ist jedoch wohlvertraut. Diese benutzt man stets, wenn man mit Uhrzeiten rechnet. Die Aussage *Es ist gerade 7 Uhr, in acht Stunden ist es 3 Uhr nachmittags* entspricht einer Addition modulo 12, da

  $$7 + 8 \equiv 15 \equiv 3 \bmod 12.$$

- Die Tabellen [15.1] und [15.2] zeigen, dass die Menge $\mathbb{Z}_5$ abgeschlossen unter der Additionsoperation bzw. Multiplikationsoperation ist.

- Sowohl die Tabelle der Addition als auch der Multiplikation zeigen, dass die Werte der Spalten gleich denen der entsprechenden Zeilen ist. Dies stellt nichts anderes dar als die Kommutativität der Operationen Addition modulo 5 und Multiplikation modulo 5. Das gilt natürlich für einen beliebigen Modulus.

- Aus der Additionstabelle [15.1] liest man ab, dass

  $$a + 0 \equiv a \bmod 5$$

  für alle $a \in \mathbb{Z}_5$. Das bedeutet, die Restklasse 0 ist das neutrale Element der Additionsoperation modulo 5.

- Aus dem Ergebnis

$$3 + 2 \equiv 0 \bmod 5$$

resultiert, dass das additive Inverse von $3 \in \mathbb{Z}_5$ die Zahl $2 \in \mathbb{Z}_5$ ist und umgekehrt. Analoges gilt für die beiden Zahlen 1 und 4.

- Die Multiplikationstabelle [15.2] zeigt, dass die Restklasse 1 das neutrale Element der Multiplikation modulo 5 ist, da

$$a \cdot 1 \equiv a \bmod 5$$

für alle $a \in \mathbb{Z}_5$ gilt.

- Aus der Tabelle [15.2] liest man die *multiplikativen Inversen* ab, da

$$\begin{aligned} 1 \cdot 1 &\equiv 1 \bmod 5 \\ 2 \cdot 3 &\equiv 1 \bmod 5 \\ 3 \cdot 2 &\equiv 1 \bmod 5 \\ 4 \cdot 4 &\equiv 1 \bmod 5 \end{aligned}$$

erhält man:

$$\begin{aligned} 1^{-1} &\equiv 1 \bmod 5 \\ 2^{-1} &\equiv 3 \bmod 5 \\ 3^{-1} &\equiv 2 \bmod 5 \\ 4^{-1} &\equiv 4 \bmod 5 \end{aligned}$$

Insbesondere haben wir die folgende Eigenschaft:
Zu jedem $a \in \mathbb{Z}_5 \setminus \{0\}$ existiert ein multiplikatives Inverses $a^{-1} \in \mathbb{Z}_5 \setminus \{0\}$ mit

$$a \cdot a^{-1} \equiv 1 \bmod 5.$$

Diese Betrachtungen zeigen, dass die Menge $\mathbb{Z}_5$ mit der modularen Addition und der modularen Multiplikation die algebraische Struktur eines Körpers hat. Da die zugrundeliegende Menge endlich viele Elemente hat — nämlich die fünf Restklassen $[0], \ldots, [4]$ — spricht man von einem **endlichen Körper**.[2] Diese Eigenschaft von $\mathbb{Z}_5$ ermöglicht es, dass man auf $\mathbb{Z}_5$ rechnen kann[3] wie mit reellen Zahlen.

Beispiel [15.30] Wir betrachten die Menge $\mathbb{Z}_2 = \{0, 1\}$
In diesem zweiten Beispiel betrachten wir die — den Informatikern vertraute — Menge $\mathbb{Z}_2$. Die Addition und Multiplikation modulo 2 werden durch folgende Tabellen beschrieben

+	**0**	**1**
0	0	1
1	1	0

×	**0**	**1**
0	0	0
1	0	1

Tabelle 15.3: Addition und Multiplikation modulo 2.

Anmerkungen:

- Aus diesen Tabellen erkennt man leicht, dass die Addition modulo 2 äquivalent zur BOOLEschen `XOR` Verknüpfung und die Multiplikation modulo 2 äquivalent zur BOOLEsche `AND` Operation ist.
- Da 0 das neutrale Element der Addition auf der Menge $\mathbb{Z}_2$ ist, ist die Addition das Gleiche wie die Subtraktion, denn

$$0 + 0 \equiv 0 \bmod 2$$
$$1 + 1 \equiv 0 \bmod 2,$$

  oder

$$a + a \equiv 0 \bmod 2 \quad \forall\, a \in \mathbb{Z}_2.$$

  Diese Eigenschaft macht man sich beispielsweise bei der Datenrekonstruktion in RAID Systemen zunutze.

Die Additions– und Multiplikationstabelle zeigen, dass auch die Menge $\mathbb{Z}_2$ mit der modularen Addition und Multiplikation die algebraische Struktur eine endlichen Körpers hat.

Beispiel [15.31]

$\mathbb{Z}_9$

Im dritten Beispiel erstellen wir die Additions– und Multiplikationstabellen der Zahlenmenge

$$\mathbb{Z}_9 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}.$$

Aus der Tabelle [15.5] erkennt man, dass in $\mathbb{Z}_9$ die Situation vorliegt, dass zwei nicht–verschwindende Elemente aus $\mathbb{Z}_9$ multipliziert werden, das Ergebnis ist 0. Betrachte beispielweise

$$3 \cdot 6 = 18 \equiv 0 \bmod 9.$$

[2] Engl.: *finite field*.
[3] Mittels Modularer Arithmetik, versteht sich.

+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8	0
2	2	3	4	5	6	7	8	0	1
3	3	4	5	6	7	8	0	1	2
4	4	5	6	7	8	0	1	2	3
5	5	6	7	8	0	1	2	3	4
6	6	7	8	0	1	2	3	4	5
7	7	8	0	1	2	3	4	5	6
8	8	0	1	2	3	4	5	6	7

Tabelle 15.4: Addition auf $\mathbb{Z}_9$.

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8
2	0	2	4	6	8	1	3	5	7
3	0	3	6	0	3	6	0	3	6
4	0	4	8	3	7	2	6	1	5
5	0	5	1	6	2	7	3	8	4
6	0	6	3	0	6	3	0	6	3
7	0	7	5	3	1	8	6	4	2
8	0	8	7	6	5	4	3	2	1

Tabelle 15.5: Multiplikation auf der Menge $\mathbb{Z}_9$.

Multiplizieren wir daher 3 und 6 mod 9 erhalten wir 0. Insbesondere liegt hier der Fall vor, dass *nicht* für jedes Element $a \in \mathbb{Z}_9 \backslash \{0\}$ ein multiplikatives Inverses $a^{-1} \in \mathbb{Z}_9 \setminus \{0\}$ existiert, so dass gilt:

$$a \cdot a^{-1} \equiv 1 \bmod 9.$$

Mit anderen Worten, die Menge $\mathbb{Z}_9$ mit der Addition und Multiplikation modulo 9 hat keine algebraische Struktur eines Körpers, oder Division ist im allgemeinen aus $\mathbf{Z}_9$ nicht definiert.

Die Multiplikationstabelle [15.5] zeigt jedoch, dass es in $\mathbb{Z}_9$ durchaus Zahlen gibt mit der Eigenschaft

$$a \cdot a^{-1} \equiv 1 \bmod 9,$$

das sind die Zahlen $1, 2, 4, 5, 7$ und 8. Es gilt:

$$\begin{aligned} 1 \cdot 1 &\equiv 1 \bmod 9, \\ 2 \cdot 5 &\equiv 1 \bmod 9, \\ 4 \cdot 7 &\equiv 1 \bmod 9, \\ 8 \cdot 8 &\equiv 1 \bmod 9. \end{aligned}$$

Beschränkt man sich daher auf die Teilmenge $\{1, 2, 4, 5, 7, 8\} \subset \mathbb{Z}_9$, findet man für jedes Element ein multiplikatives Inverses.

15.3 Elementare Eigenschaften von Restklassen

15.3.1 Modulare Arithmetik

Der folgende Satz stellt die Grundlage der **Modularen Arithmetik** dar.

Theorem [8]:

Sei

$$a \equiv b \mod n$$
$$c \equiv d \mod n$$

dann folgt

1. $a + c \equiv b + d \mod n$
2. $a \cdot c \equiv b \cdot d \mod n$
3. $k \cdot a \equiv k \cdot b \mod n$

für alle $k \in \mathbb{Z}$.

Dieser Satz sagt aus, dass man mit Kongruenzen die Additions–, Subtraktions– und Multiplikationsoperation ausführen kann, wie man es von ganzen Zahlen gewohnt ist. Die Division erfordert eine zusätzliche Eigenschaft. Diese betrachten wir daher im folgenden Abschnitt.

Beweis:

Die Aussagen des Satzes folgen direkt aus der Definition der Kongruenzrelation.

1. Wenn
$$a \equiv b \mod n$$
dann ist n Teiler von $a - b$, *i.e.* $n \mid (a - b)$. Falls
$$c \equiv d \mod n$$
dann ist n auch Teiler von $c - d$, *i.e.* $n \mid (c - d)$. Daher ist n Teiler der Summe dieser beiden Differenzen, *i.e.* $n \mid (a - b + c - d)$ oder $n \mid ((a + c) - (b + d))$. Dies bedeutet aber
$$a + c \equiv b + d \mod n.$$

2. Wenn n Teiler von $a-b$ und $c-d$ ist, dann ist n auch Teiler des Produktes. Also, da $n \mid (a - b)$ und $n \mid (c - d)$ dann ist auch
$$n \mid (a - b) \cdot (c - d).$$
Nun ist
$$(a - b) \cdot (c - d) = a \cdot c - b \cdot d + b \cdot (d - c) + d \cdot (b - a).$$

Da n Teiler des dritten und vierten Terms auf der rechten Seite ist, dann ist auch $n \mid (a \cdot c - b \cdot d)$ *i.e.*

$$a \cdot c \equiv b \cdot d \mod n.$$

3. Schließlich, falls $n \mid (a - b)$ dann gilt offensichtlich auch $n \mid k \cdot (a - b)$ für ein beliebiges $k \in \mathbb{Z}$.

Beispiel [15.32]

Betrachte die beiden Kongruenzen

$$146 \equiv 11 \bmod 15,$$
$$23 \equiv 8 \bmod 15.$$

In der einen Richtung addieren wir die beiden linken Seiten und bilden anschließend den Rest. Wir erhalten

$$146 + 23 = 169 \equiv 4 \bmod 15.$$

Andererseits können wir gemäß obigem Satz erst die Reste bilden, die Reste addieren und nochmals den Rest bilden:

$$11 + 8 = 19 \equiv 4 \bmod 15$$

Wichtig zu erkennen ist, dass durch die Anwendung dieses Satzes bei der Addition modulo n die die zu addierenden Zahlen stets klein bleiben.

Noch deutlicher wird der Vorteil mit Resten zu arbeiten bei der Multiplikation. Betrachte die folgenden Kongruenzen:

$$143 \equiv 3 \bmod 7$$
$$36 \equiv 1 \bmod 7$$

Wir bilden zuerst das Produkt der linken Seiten und anschließend den Rest.

$$143 \cdot 36 = 5148 \equiv 3 \bmod 7$$

Anwenden des obigen Satzes bedeutet, daß zuerst die Reste gebildet werden und anschließend multiplizieren:

$$3 \cdot 1 = 3 \equiv 3 \bmod 7$$

Beispiel [15.33]

Die FERMAT–Zahlen sind definiert durch:

$$F_n = 2^{2^n} + 1.$$

FERMAT lag richtig mit der Annahme, dass die ersten vier dieser Zahlen Primzahlen sind. Die Frage ist, ob die fünfte FERMAT Zahl

$$F_5 = 2^{2^5} + 1 = 2^{32} + 1 = 4\,294\,967\,297$$

ebenfalls Primzahl ist. Wir zeigen nun mit Hilfe der modularen Arithmetik, dass FERMAT mit seiner Vermutung etwas daneben lag, und sehen, dass 641 ein Teiler dieser Zahl ist.

Zunächst ist

$$641 = 640 + 1 = 5 \cdot 2^7 + 1.$$

Dies hat zur Folge, dass

$$5 \cdot 2^7 \equiv -1 \bmod 641.$$

Mit Hilfe der zweiten Aussage des Satzes [8] erhalten wir:

$$(5 \cdot 2^7)^4 \equiv (-1)^4 \bmod 641. \tag{15.3}$$

Dies folgt einfach aus der Tatsache, wenn

$$a \equiv b \bmod n$$

dann gilt auch

$$a^2 \equiv b^2 \bmod n$$

usw. Daher haben wir:

$$5^4 \cdot 2^{28} \equiv 1 \bmod 641$$

Weiterhin:

$$641 = 625 + 16 = 5^4 + 2^4$$

das heißt:

$$5^4 \equiv (-2)^4 \bmod 641.$$

Daher können wir in der Kongruenz 15.3 5^4 durch $(-2)^4$ ersetzen:

$$(-2)^{32} \equiv 1 \bmod 641,$$

und damit

$$2^{2^5} + 1 \equiv 0 \bmod 641,$$

was zur Folge hat: $641 \mid F_5$.

Beispiel [15.34]

Der Satz [8] ermöglicht die Berechnung von Kongruenzen ohne mit großen Zahlen arbeiten zu müssen. Wenn man $a = b$ in der zweiten Aussage des obigen Satzes setzt, erhält man

$$a \cdot a = a^2 \equiv b \cdot b \bmod n = b^2 \bmod n$$

Wiederholte Anwendung dieser Eigenschaft der Kongruenzen führt auf folgende Regel: Für beliebiges $m \in \mathbb{N}$ gilt

$$a^m \equiv b^m \bmod n.$$

Um zu demonstrieren, wie dies funktioniert, betrachten wir:

$$11 \equiv 2 \bmod 9$$

Gemäß dem obigen Satz gilt daher

$$11^2 \equiv 4 \bmod 9$$

Da $11^2 = 121$ und 121 dividiert durch 9 ist 13 Rest 4 ist also in der Tat

$$121 \equiv 4 \bmod 9$$

Beispiel [15.35]

Mit den Rechenregeln Satzes [8] lassen sich also folgende Fragen leicht beantworten:

→ Was ist der Rest, wenn man $73 \cdot 52$ durch 7 teilt?

→ Ist die Zahl $(2^{15}) \cdot (14^{40}) + 1$ durch 11 teilbar?

Um die erste Frage zu beantworten, können wir die linke Seite erst komplett ausmultiplizieren, $73 \times 52 = 3796$ und dieses Produkt durch 7 teilen, was auf den Rest 2 führt. Aufgrund des obigen Satzes besteht aber keine Notwendigkeit, dies zu tun. Da

$$\begin{aligned} 73 &\equiv 3 \bmod 7 \\ 52 &\equiv 3 \bmod 7 \end{aligned}$$

finden wir

$$73 \cdot 52 \equiv 3 \cdot 3 \bmod 7 \equiv 9 \bmod 7 \equiv 2 \bmod 7.$$

Die zweite Frage kann folgendermaßen beantwortet werden. Da

$$\begin{aligned} 2^5 = 32 &\equiv -1(\bmod 11) \\ 14^2 \equiv 3^2 \bmod 11 &\equiv -2 \bmod 11 \end{aligned}$$

folgt

$$\begin{aligned} (2^{15}) \cdot (14^{40}) &\equiv (2^5)^3 \cdot (3^2)^{20} \bmod 11 \\ &\equiv (-1)^3 \cdot (-2)^{20} \bmod 11 \\ &\equiv (-1) \cdot (2)^{20} \bmod 11 \\ &\equiv (-1) \cdot (2^5)^4 \bmod 11 \\ &\equiv (-1) \cdot (-1)^4 \bmod 11 \\ &\equiv -1 \bmod 11 \end{aligned}$$

Daher ist $(2^{15}) \cdot (14^{40}) + 1 \equiv 0 \bmod 11$, *i.e.* die Zahl ist durch 11 teilbar.

Behauptung:

Die modulare Arithmetik hat folgende Eigenschaften:

$$[(a(\bmod n)) + (b(\bmod n))](\bmod n) \equiv (a+b) \bmod n \quad (15.4)$$
$$[(a(\bmod n)) - (b(\bmod n))](\bmod n) \equiv (a-b) \bmod n \quad (15.5)$$
$$[(a(\bmod n)) \cdot (b(\bmod n))](\bmod n) \equiv (a \cdot b) \bmod n \quad (15.6)$$

Beweis:

Wir definieren zunächst:

$$a \bmod n = r_a$$
$$b \bmod n = r_b$$

so daß

$$a = r_a + k \cdot n$$
$$b = r_b + l \cdot n$$

für irgendwelche Zahlen $k, l \in \mathbb{Z}$. Dann ist:

$$\begin{aligned}(a+b)(\bmod n) &= (r_a + k \cdot n + r_b + l \cdot n)(\bmod n)) \\ &= (r_a + r_b + (k+l) \cdot n)(\bmod n)) \\ &= (r_a + r_b)(\bmod n)) \\ &= [(a(\bmod n)) + (b(\bmod n))](\bmod n)\end{aligned}$$

Die anderen Eigenschaften zeigt man auf ähnliche Weise.

Beispiel [15.36]

① Betrachte

$$11 \bmod 8 = 3 \quad \text{und} \quad 15 \bmod 8 = 7;$$

Dann folgt:

$$[(11 \bmod 8) + (15 \bmod 8)] \bmod 8 = 10 \bmod 8 = 2$$

Die andere Richtung ergibt:

$$(11 + 15) \bmod 8 = 26 \bmod 8 = 2$$

② Betrachte:

$$11 \bmod 8 = 3 \quad \text{und} \quad 15 \bmod 8 = 7;$$

Dann ist die Differenz:

$$[(11 \bmod 8) - (15 \bmod 8)] \bmod 8 = -4 \bmod 8 = 4$$

und

$$(11 - 15) \bmod 8 = -4 \bmod 8 = 4$$

③ Schließlich ist das Produkt:

$$[(11 \bmod 8) \cdot (15 \bmod 8)] \bmod 8 = 21 \bmod 8 = 5$$

und die andere Richtung:

$$(11 \cdot 15) \bmod 8 = 165 \bmod 8 = 5$$

15.3.2 Division

Es gibt eine Feinheit in der modularen Arithmetik gegenüber der Arithmetik mit ganzen Zahlen, die die Division betrifft. Zunächst: Wie in der gewöhnlichen Arithmetik gilt in der modularen Arithmetik:

Falls

$$(a + b) \equiv (a + c) \bmod n,$$

dann gilt

$$b \equiv c \bmod n$$

Dies zeigt man folgendermaßen: Sei

$$(a + b) \equiv (a + c) \bmod n.$$

Die impliziert per Definition der Kongruenz, dass n Teiler der Differenz $(a + b) - (a + c) = b - c$ ist. Da dann n auch Teiler von $b - c$ ist, haben wir in der Tat

$$b \equiv c \bmod n$$

Die *Division modulo* n gilt jedoch nur mit einer zusätzlichen Bedingung.

Behauptung:

Seien a, b, c, n ganze Zahlen, $n \neq 0$ und $\texttt{ggT}\ (a, n) = 1$. Wenn

$$(a \cdot b) \equiv (a \cdot c) \bmod n$$

dann ist

$$b \equiv c \bmod n$$

Mit anderen Worten, nur dann, wenn a und n coprim sind, kann eine Kongruenz durch a dividiert werden.

Beweis:

Da wir annehmen, dass $\texttt{ggT}\ (a, n) = 1$, gibt es gemäß dem Lemma von BÉZOUT zwei ganze Zahlen $k, l \in \mathbb{Z}$ so dass

$$a \cdot k + n \cdot l = 1.$$

Multiplizieren wir diese Gleichung mit $b - c$, erhalten wir:

$$(ab - ac) \cdot k + n(b - c) \cdot l = b - c$$

Da gemäß Annahme $ab \equiv ac \bmod n$ gilt, ist $ab - ac$ ein Vielfaches von n; der zweite Term $n(b - c) \cdot l$ ist sicherlich ein Vielfaches von n, daher schließen wir, dass die rechte Seite dieser Gleichung, *i.e.* der Term $b - c$ ebenfalls ein Vielfaches von n sein muß. Das heißt aber, dass $b \equiv c \bmod n$ gilt.

Beispiel [15.37]

① Löse die Gleichung

$$2x + 7 \equiv 3 \bmod 17$$

nach x auf.

Lösung:

$$\begin{aligned} 2x &\equiv 3 - 7 \bmod 17 \\ &\equiv -4 \bmod 17, \end{aligned}$$

daher

$$x \equiv -2 \equiv 15 \bmod 17.$$

Die Division durch 2 ist erlaubt, da, $\texttt{ggT}\ (2, 17) = 1$.

Check:

$$2 \cdot 15 + 7 \bmod 17 \equiv 37 \bmod 17 \equiv 3 \bmod 17.$$

② Löse die Gleichung

$$5x + 6 \equiv 13 \bmod 11$$

nach x auf.

Lösung:

$$5x \equiv 7 \bmod 11$$

Wie kann nun weiter verfahren werden? Das Ziel ist, diese Gleichung durch 5 zu teilen. Dies ist erlaubt, da die Zahlen 5 und 11 sicherlich coprim sind. Die Frage ist daher, welche Bedeutung ein Ausdruck der Form 7/5 mod 11 hat.

Wir erinnern uns, dass wir mit Restklassen arbeiten, daher kann ein beliebiger Vertreter der Äquivalenzklasse benutzt werden. Diese möglichen Repräsentanten sind z.B.:

$$7 \equiv 18 \equiv 29 \equiv 40 \equiv \ldots \bmod 11$$

Daher ist $5x \equiv 7 \bmod 11$ gleichwertig mit $5x \equiv 40 \bmod 11$. In diesem Fall können wir durch 5 teilen und erhalten die Antwort

$$x \equiv 8 \bmod 11$$

Check:

$$5 \cdot 8 + 6 \bmod 13 \equiv 46 \bmod 13 \equiv 7 \bmod 13.$$

15.3.3 Modulare Exponentiation

Wenn man mit kryptographischen Algorithmen arbeitet — insbesondere bei der Public Key Kryptographie — hat man häufig Zahlen der Form[4]

$$z = x^a \bmod n$$

zu berechnen. Angenommen, man möchte die Zahl

$$z = 2^{1234} \bmod 789$$

berechnen. Wenn man zunächst die Zahl 2^{1234} berechnet und anschließend die Zahl mod 789 reduziert, muß man mit sehr großen Zahlen arbeiten, obwohl sich das Endresultat auf eine dreistellige Zahl reduziert. Die adq̈uate Methode, solche Ausdrücke zu berechnen besteht daher darin, die modulo Reduktion geschickt auszunutzen.

Das folgende Verfahren ist eine sehr effektive Methode, um mit großen Zahlen umzugehen. Wir demonstrieren dieses Verfahren zunächst zur Berechnung von

$$z = 2^{1234} \bmod 789$$

[4] Dieser Abschnitt ist aus [215]; siehe auch [218].

Wir beginnen mit der Berechnung von $2^2 \equiv 4 \bmod 789$ und quadrieren beide Seiten sukzessive. Dabei erhalten wir die folgende Reihe von Kongruenzen:

$$\begin{aligned}
2^2 &\equiv 4 \bmod 789 \\
2^4 &\equiv 4^2 \bmod 789 \equiv 16 \bmod 789 \\
2^8 &\equiv 16^2 \bmod 789 \equiv 256 \bmod 789 \\
2^{16} &\equiv 256^2 \bmod 789 \equiv 49 \bmod 789 \\
2^{32} &\equiv 49^2 \bmod 789 \equiv 34 \bmod 789 \\
2^{64} &\equiv 34^2 \bmod 789 \equiv 367 \bmod 789 \\
2^{128} &\equiv 367^2 \bmod 789 \equiv 559 \bmod 789 \\
2^{256} &\equiv 559^2 \bmod 789 \equiv 37 \bmod 789 \\
2^{512} &\equiv 37^2 \bmod 789 \equiv 580 \bmod 789 \\
2^{1024} &\equiv 580^2 \bmod 789 \equiv 286 \bmod 789
\end{aligned}$$

Nun kann der Exponent, *i.e.* die Zahl 1234 als Summe von Zweierpotenzen in folgender Weise geschrieben werden:

$$1234 = 1024 + 128 + 64 + 16 + 2$$

i.e.:

$$1234_d = 10\,011\,010\,010_2$$

Wir machen nun mehrfach Gebrauch von der Eigenschaft:

$$(a \cdot b) \bmod n \equiv [(a \bmod n) \cdot (b \bmod n)] \bmod n$$

und finden:

$$\begin{aligned}
2^{1234} \bmod 789 &= 2^{1024+128+64+16+2} \bmod 789 \\
&= [2^{1024} \cdot 2^{128} \cdot 2^{64} \cdot 2^{16} \cdot 2^2] \bmod 789 \\
&\equiv \Big[(2^{1024} \bmod n) \cdot (2^{128} \bmod n) \cdot (2^{64} \bmod n) \cdot \\
&\quad (2^{16} \bmod n) \cdot (2^2 \bmod n)\Big] \bmod 789 \\
&\equiv \Big[286 \cdot 559 \cdot 367 \cdot 49 \cdot 4\Big] \bmod 789 \\
&\equiv 481 \bmod 789
\end{aligned}$$

Mit dieser Anwendung der modularen Arithmetik müssen wir niemals mit Zahlen größer als 788^2 arbeiten.

Dieses Verfahren kann verallgemeinert werden zu einem Algorithmus. Falls eine Zahl der Form

$$z = a^b \bmod n$$

berechnet werden muß, kann dies durchgeführt werden mit höchstens $2 \log_2(b)$ Multiplikationen modulo n. Man muß niemals mit Zahlen größer als n^2 arbeiten. Dies impliziert, dass modulare Exponentiation sehr effektiv implementiert werden kann und man benötigt nicht extrem viel Speicherplatz.

Das Verfahren der modularen Exponentiation ist sehr nützlich, wenn a, b und n sehr große Zahlen sind, *e.g.* 100stellige Dezimalzahlen, was in aktuellen kryptographischen Verfahren die Regel ist. Berechnet man erst a^b und reduziert anschließend modulo n, führt dies unvermeidlich auf einen Speicherüberlauf. Denn die Zahl a^b hat mehr als 10^{100} Stellen, das heißt mehr Stellen als es Atome im Universum gibt. Die Berechnung von $a^b \bmod n$ kann jedoch mit dem vorgestellten Verfahren in weniger als 700 Schritten durchgeführt werden, ohne mit Zahlen arbeiten zu müssen, die mehr als 200 Stellen haben.

Algorithmus Modulare Exponentiation

Der folgende Algorithmus berechnet `fastexp`$= x^y \bmod n$

ME 0 *Input:* Drei positive Zahlen x, y, n, drei Hilfsariablen a, b, c.

ME 1 *Initialisierung:* Setze:

$$\begin{aligned} a &\longleftarrow x \\ b &\longleftarrow 1 \\ c &\longleftarrow y \end{aligned}$$

ME 2 Falls c gerade ist, setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n \\ b &\longleftarrow b \\ c &\longleftarrow c/2 \end{aligned}$$

ME 3 Falls c ungerade ist, setze

$$\begin{aligned} a &\longleftarrow a \\ b &\longleftarrow (b \cdot a) \bmod n \\ c &\longleftarrow c - 1 \end{aligned}$$

ME 4 Falls $c \neq 0$ gehe zu Schritt **ME 2**, sonst `STOP`; output ist $b =$ `fastexp`.

Beispiel [15.38]

① Sehen wir uns nun anhand von konkreten Beispielen an, wie dieser Algorithmus arbeitet. Dazu arbeiten wir zunächst ein einfaches Beispiel aus. Wir berechnen

$$z = x^y \bmod n = 7^9 \bmod 5.$$

Mit dem Taschenrechner überzeugt man sich schnell, dass das Ergebnis 2 sein muß.

Wendet man die obige Papier–und–Bleistift–Methode an, wird zunächst

der Ausdruck 7^9 zerlegt gemäß:

$$7^9 \bmod 5 \equiv 7^{8+1} \bmod 5 \equiv [(7^8 \bmod 5) \cdot (7 \bmod 5)] \bmod 5.$$

Mit

$$\begin{aligned} 7^2 \bmod 5 &\equiv 49 \bmod 5 \equiv 4 \bmod 5 \\ 7^4 \bmod 5 &\equiv 16 \bmod 5 \equiv 1 \bmod 5 \\ 7^8 \bmod 5 &\equiv 1 \bmod 5 \end{aligned}$$

erhält man in der Tat:

$$7^9 \bmod 5 \equiv 7^{8+1} \bmod 5 \equiv [1 \cdot 75] \bmod 5 \equiv 2 \bmod 5.$$

Wir wenden nun den Algorithmus Modulare Exponentiation an, um dieses Resultat zu verifizieren:

Step 1: **ME 0** Input: $x = 7, y = 9, n = 5$ und drei Hilfsvariable a, b, c.

Step 2: **ME 1** Initialisierung: Wir setzen

$$\begin{aligned} a &\longleftarrow x = 7 \\ b &\longleftarrow 1 \\ c &\longleftarrow y = 9 \end{aligned}$$

Step 3: **ME 2** c gerade? Nein; weiter mit Schritt: **ME 3**.

Step 4: **ME 3** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a = 7 \\ b &\longleftarrow (b \cdot a) \bmod n = (1 \cdot 7) \bmod 5 \equiv 2 \\ c &\longleftarrow c - 1 = 8 \end{aligned}$$

Step 5: **ME 4** $c \neq 0$ Ja; weiter mit Schritt: **ME 2**.

Step 6: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 7^2 \bmod 5 \equiv 4 \bmod 5 \\ b &\longleftarrow b = 2 \\ c &\longleftarrow c/2 = 4 \end{aligned}$$

Step 7: **ME 4** $c \neq 0$ Ja; weiter mit Schritt: **ME 2**.

Step 8: **ME 2** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 4^2 \bmod 5 \equiv 1 \bmod 5 \\ b &\longleftarrow b = 2 \\ c &\longleftarrow c/2 = 2 \end{aligned}$$

Step 9: **ME 4** $c \neq 0$ Ja; weiter mit Schritt: **ME 2**.

Step 10: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 1^2 \bmod 5 \equiv 1 \bmod 5 \\ b &\longleftarrow b = 2 \\ c &\longleftarrow c/2 = 1 \end{aligned}$$

Step 11: **ME 4** $c \neq 0$ Ja; weiter mit Schritt: **ME 2**.

Step 12: **ME 2** c gerade? Nein; weiter mit **ME 3**.

Step 13: **ME 3** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a = 1 \\ b &\longleftarrow (b \cdot a) \bmod n = (2 \cdot 1) \bmod 5 \equiv 2 \\ c &\longleftarrow c - 1 = 0 \end{aligned}$$

Step 14: **ME 4** $c \neq 0$ Nein, `STOP`; output ist `fastexp` $= b = 2$.

② Im zweiten Beispiel wiederholen wir die nichttriviale Berechnung von

$$z = 2^{1234} \bmod 789$$

mit Hilfe des Algorithmus Modulare Exponentiation.

Step 1: **ME 0** Input: $x = 2, y = 1234, n = 789$ und drei Hilfsvariable a, b, c.

Step 2: **ME 1** Initialisierung: Setze

$$\begin{aligned} a &\longleftarrow x = 2 \\ b &\longleftarrow 1 \\ c &\longleftarrow y = 1234 \end{aligned}$$

Step 3: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 4 \bmod 789 \\ b &\longleftarrow b = 1 \\ c &\longleftarrow c/2 = 617 \end{aligned}$$

Step 4: **ME 4** $c \neq 0$ Ja; weiter mit Schritt **ME 2**.

Step 5: **ME 2** c gerade? Nein; weiter mit **ME3**

Step 6: **ME 3** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a = 4 \\ b &\longleftarrow (b \cdot a) \bmod n = (1 \cdot 4) \bmod 789 \equiv 4 \\ c &\longleftarrow c - 1 = 616 \end{aligned}$$

Step 7: **ME 4** $c \neq 0$ Ja; weiter mit Schritt **ME 2**.

Step 8: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 16 \bmod 789 \\ b &\longleftarrow b = 4 \\ c &\longleftarrow c/2 = 308. \end{aligned}$$

Step 9: **ME 4** $c \neq 0$ Ja, weiter mit Schritt **ME 2**.

Step 10: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 256 \bmod 789 \\ b &\longleftarrow b = 4 \\ c &\longleftarrow c/2 = 154. \end{aligned}$$

Step 11: **ME 4** $c \neq 0$ Ja; weiter mit **ME 2**.

Step 12: **ME 2** c gerade? Ja, setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 256^2 \bmod 789 \equiv 49 \bmod 789 \\ b &\longleftarrow b = 4 \\ c &\longleftarrow c/2 = 77. \end{aligned}$$

Step 13: **ME 4** $c \neq 0$ Ja; weiter mit **ME 2**.

Step 14: **ME 2** c gerade? Nein; weiter mit **ME3**

Step 15: **ME 3** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a = 49 \\ b &\longleftarrow (b \cdot a) \bmod n = (4 \cdot 49) \bmod 789 \equiv 196 \bmod 789 \\ c &\longleftarrow c - 1 = 76. \end{aligned}$$

Step 16: **ME 4** $c \neq 0$ Ja; weiter mit **ME 2**.

Step 17: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 49^2 \bmod 789 \equiv 34 \bmod 789 \\ b &\longleftarrow b = 196 \\ c &\longleftarrow c/2 = 38. \end{aligned}$$

Step 18: **ME 4** $c \neq 0$ Ja; weiter mit **ME 2**.

Step 19: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 34^2 \bmod 789 \equiv 367 \bmod 789 \\ b &\longleftarrow b = 196 \\ c &\longleftarrow c/2 = 19. \end{aligned}$$

Step 20: **ME 4** $c \neq 0$ Ja; weiter mit **ME 2**.

Step 21: **ME 2** c gerade? Nein; weiter mit **ME3**.

Step 22: **ME 3** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a = 367 \\ b &\longleftarrow (b \cdot a) \bmod n = (196 \cdot 367) \bmod 789 \equiv 133 \bmod 789 \\ c &\longleftarrow c - 1 = 18. \end{aligned}$$

Step 23: **ME 4** $c \neq 0$ Ja; weiter mit Schritt **ME 2**.

Step 24: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 367^2 \bmod 789 \equiv 559 \bmod 789 \\ b &\longleftarrow b = 133 \\ c &\longleftarrow c/2 = 9. \end{aligned}$$

Step 25: **ME 4** $c \neq 0$ Ja; weiter mit Schritt **ME 2**.

Step 26: **ME 2** c gerade? Nein; weiter mit Schritt **ME3**.

Step 27: **ME 3** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a = 559 \\ b &\longleftarrow (b \cdot a) \bmod n = (133 \cdot 559) \bmod 789 \equiv 181 \bmod 789 \\ c &\longleftarrow c - 1 = 8. \end{aligned}$$

Step 28: **ME 4** $c \neq 0$ Ja; weiter mit **ME 2**.

Step 29: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 559^2 \bmod 789 \equiv 37 \bmod 789 \\ b &\longleftarrow b = 181 \\ c &\longleftarrow c/2 = 4. \end{aligned}$$

Step 30: **ME 4** $c \neq 0$ Ja; weiter mit Schritt **ME 2**.

Step 31: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 37^2 \bmod 789 \equiv 580 \bmod 789 \\ b &\longleftarrow b = 181 \\ c &\longleftarrow c/2 = 2. \end{aligned}$$

Step 32: **ME 4** $c \neq 0$ Ja; weiter mit **ME 2**.

Step 33: **ME 2** c gerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a^2 \bmod n = 580^2 \bmod 789 \equiv 286 \bmod 789 \\ b &\longleftarrow b = 181 \\ c &\longleftarrow c/2 = 1. \end{aligned}$$

Step 34: **ME 4** $c \neq 0$? Ja; weiter mit **ME 2**.

Step 35: **ME 2** c gerade? Nein; weiter mit Schritt **ME3**.

Step 36: **ME 3** c ungerade? Ja; setze

$$\begin{aligned} a &\longleftarrow a = 286 \\ b &\longleftarrow (b \cdot a) \bmod n = (181 \cdot 286) \bmod 789 \equiv 481 \bmod 789 \\ c &\longleftarrow c - 1 = 0. \end{aligned}$$

Step 37: **ME 4** $c \neq 0$? Nein; `STOP`; output ist `fastexp` $= b = 481$.

Wir haben daher das obige Resultat reproduziert.

Nachdem wir die Arbeitsweise des Algorithmus anhand von Beispielen durchexerziert haben, müssen wir uns dem schwierigeren Teil widmen, *i.e.* wir haben den folgenden Satz zu beweisen:

Theorem [9]:

Der Algorithmus Modulare Exponentiation terminiert und liefert das Resultat $x^y \bmod n$.

Beweis:

Es sind zwei Aussagen zu beweisen:

❶ Der Algorithmus terminiert.

❷ Der Algorithmus ist korrekt.

❶ Der Algorithmus terminiert, wenn $c = 0$. Da jeder Zwischenschritt des Algorithmus den Wert von c reduziert, tritt dieser Fall $c = 0$ immer ein. Daher terminiert dieser Algorithmus in allen Fällen.

❷ Die Korrektheit des Algorithmus zeigen wir durch mathematische Induktion. Die Schleifeninvariante des `fastexp` Algorithmus ist

$$b \cdot a^c \bmod n = x^y \bmod n$$

(a) *Induktionsbeginn:* Beim erstmaligen Eintritt in den Loop gilt:

$$a \leftarrow x$$
$$b \leftarrow 1$$
$$c \leftarrow y$$

Daher ist

$$b \cdot a^c \bmod n = 1 \cdot x^y \bmod n$$

(b) *Induktionsannahme:* Der Algorithmus hat m Runden durchlaufen und vor dem Eintritt in die Runde $m+1$ haben die Variablen folgende Werte:

$$a \leftarrow a'$$
$$b \leftarrow b'$$
$$c \leftarrow c'$$

und die Induktionsannahme lautet

$$b' \cdot (a')^{c'} \bmod n = x^y \bmod n$$

(c) *Induktionsschluß:* Die Schleife wird zum $m+1$ten Mal durchlaufen. Dabei können zwei Fälle auftreten:

Fall a: c' ist gerade. Dann haben die Variablen nach dem Durchlauf die Werte:

$$\begin{aligned} a &\leftarrow (a')^2 \bmod n \\ b &\leftarrow b' \\ c &\leftarrow c'/2 \end{aligned}$$

und es folgt:

$$\begin{aligned} b \cdot a^c \bmod n &= b' \cdot ((a')^2)^{c'/2} \bmod n \\ &= b' \cdot (a')^{c'} \bmod n \\ &= x^y \bmod n \end{aligned}$$

wobei im letzten Schritt die Induktionsannahme eingeht.

Fall b: c' ist ungerade; dann gilt

$$\begin{aligned} a &\leftarrow a' \\ b &\leftarrow (b' \cdot a') \bmod \mathrm{n} \\ c &\leftarrow c' - 1 \end{aligned}$$

und es folgt in diesem Fall:

$$\begin{aligned} b \cdot a^c \bmod n &= (b' \cdot a') \cdot (a')^{c'-1} \bmod n \\ &= b' \cdot (a')^{c'} \bmod n \\ &= x^y \bmod n \end{aligned}$$

wobei im letzten Schritt die Induktionsannahme eingeht.

(d) Der Abbruch der Schleife erfolgt mit $c = 0$. Damit liefert der Algorithmus die Ausgabe

$$\begin{aligned} b &= b \cdot a^0 \bmod n \\ &= x^y \bmod n \end{aligned}$$

15.4 Der Chinesische Restesatz

Wir betrachten zunächst das folgende Beispiel:

Beispiel [15.39]

Ein bayerischer Bauer möchte seine Kühe stolz auf dem Volksfestzug präsentieren. Wenn er seine Kühe in 3er–Reihen aufstellt, bleiben 2 Kühe übrig. Stellt er sie in 4er–Reihen auf, bleibt eine Kuh übrig. Erst wenn er die Kuhherde in 7er–Reihen aufstellt, bleibt keine Kuh mehr übrig. Die Frage ist, wieviele Kühe hat der Bauer?

Der **Chinesische Restsatz** sagt aus, dass eine Lösung solcher Probleme existiert.

Theorem [10]:

Gegeben sind die simultanen Kongruenzen

$$x \equiv a \bmod m, \tag{15.7}$$
$$x \equiv b \bmod n \tag{15.8}$$

mit $\mathrm{ggT}(m,n) = 1 = u \cdot m + v \cdot n$. Dann wird dieses System gelöst durch

$$x \equiv (v \cdot n \cdot a + u \cdot m \cdot b) \bmod m \cdot n. \tag{15.9}$$

Beweis

Wir betrachten die beiden Kongruenzen

$$x \equiv a \bmod m,, \tag{15.10}$$
$$x \equiv b \bmod n. \tag{15.11}$$

Aus Gl. (15.10) folgt:

$$x = t \cdot m + a. \tag{15.12}$$

Setzen wir dies in Gl. (15.11) ein, erhalten wir:

$$a + t \cdot m \equiv b \bmod n,$$

oder

$$t \cdot m \equiv b - a \bmod n. \tag{15.13}$$

Gemäß Voraussetzung sind die beiden Restklassen teilerfremd, *i.e.* es gilt

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

Gemäß dem Lemma von BÉZOUT existieren Zahlen $u, v \in \mathbb{Z}$ mit

$$\text{ggT}(m,n) = 1 = u \cdot m + v \cdot n.$$

Da m, n coprim sind, existiert das multiplikative Inverse $m^{-1} \bmod n$, wir können daher Gl. (15.13) nach t auflösen:

$$t \equiv m^{-1}(b-a) \bmod m. \tag{15.14}$$

Da $1 = u \cdot m + v \cdot n$, erhalten wir

$$u \cdot m \equiv 1 \mod n,$$

oder

$$u \equiv m^{-1} \bmod n.$$

Damit wird Gl. (15.14):

$$t \equiv u \cdot (b-a) \bmod n.$$

Wir schreien dies in der Form:

$$t = u(b-a) + l \cdot n, \qquad l \in \mathbb{Z}.$$

Ersetzen wir in Gl. (15.12) den Parameter t durch (15.14) erhalten wir:

$$\begin{aligned} x &= t \cdot m + a \\ &= (u(b-a) + l \cdot n) \cdot m + a \\ &= u \cdot (b-a) \cdot m + l \cdot m \cdot n + a \\ &= u \cdot b \cdot m - u \cdot a \cdot m + a + l \cdot m \cdot n \\ &= u \cdot b \cdot m + a\,(1 - u \cdot m) + l \cdot m \cdot n \\ &= u \cdot b \cdot m + a \cdot v \cdot n + l \cdot m \cdot n \\ &= (u \cdot b \cdot m + a \cdot v \cdot n) \bmod m \cdot n. \end{aligned}$$

Beispiel [15.40]

Wir lösen das im Beispiel [15.39] gestellte Problem. Der bayerische Landwirt steht vor dem Problem, die folgenden simultanen Kongruenzen zu lösen:

$$x \equiv 2 \bmod 3, \tag{15.15a}$$
$$x \equiv 1 \bmod 4, \tag{15.15b}$$
$$x \equiv 0 \bmod 7. \tag{15.15c}$$

Wir können dieses System sukzessive lösen. Betrachte dazu die beiden Kongruenzen

$$x \equiv 2 \bmod 3,$$
$$x \equiv 1 \bmod 4$$

Vergleichen wir diese beiden Kongruenzen mit dem System (15.7) und (15.8) dann ist

$$\begin{aligned} a &= 2, \quad m = 3, \\ b &= 1, \quad n = 4. \end{aligned}$$

Da

$$\mathrm{ggT}(m,n) = \mathrm{ggT}(3,4) = 1 = -5 \cdot 3 + 4 \cdot 4,$$

erhalten wir

$$u = -5, \qquad v = 4.$$

Setzen wir diese Parameter in die Lösung (15.9) ein, erhalten wir:

$$\begin{aligned} x &\equiv (v \cdot n \cdot a + u \cdot m \cdot b) \bmod m \cdot n \\ &= (4 \cdot 4 \cdot 2 + (-5) \cdot 3 \cdot 1) \bmod 3 \cdot 4 \\ &= 32 - 15 \bmod 12 \\ &= 17 \bmod 12 \\ &\equiv 5 \bmod 12. \end{aligned} \tag{15.16}$$

Man beachte, dass in der Tat gilt;

$$\begin{aligned} 5 &\equiv 2 \bmod 3, \\ 5 &\equiv 1 \bmod 4, \end{aligned}$$

i.e. $x = 5$ erfüllt die beiden Kongruenzen (15.15a) und (15.15b) simultan. Wir betrachten in einem zweiten Schritt die beiden simultanen Kongruenzen (15.15c) und (15.16), *i.e.*

$$\begin{aligned} x &\equiv 5 \bmod 12, \\ x &\equiv 0 \bmod 7. \end{aligned}$$

Die Parameter sind hier

$$\begin{aligned} a &= 5, \quad m = 12, \\ b &= 0, \quad n = 7. \end{aligned}$$

Es gilt auch hier

$$\mathrm{ggT}(m,n) = \mathrm{ggT}(12,5) = 1 = 3 \cdot 12 - 7 \cdot 5,$$

also haben wir

$$u = 3, \quad v = -5.$$

Setzen wir diese neuen Parameter wieder in die Lösung (15.9) ein, erhalten wir:

$$\begin{aligned} x &\equiv (v \cdot n \cdot a + u \cdot m \cdot b) \bmod m \cdot n \\ &= (-5 \cdot 7 \cdot 5 + 3 \cdot 12 \cdot 0) \bmod 12 \cdot 7 \\ &= -175 \bmod 84 \\ &\equiv 77 \bmod 84. \end{aligned}$$

Es gilt in der Tat:

$$77 \equiv 2 \bmod 3,$$
$$77 \equiv 1 \bmod 4,$$
$$77 \equiv 0 \bmod 7.$$

Das bedeutet, der Bauer hat (mindestens) 77 Kühe (oder ein Vielfaches davon).

Beispiel [15.41]

Wir suchen die Zahl x, die die folgenden drei Kongruenzen

$$x \equiv 1 \bmod 3$$
$$x \equiv 3 \bmod 7$$
$$x \equiv 5 \bmod 11$$

gleichzeitig löst. Im obigen Formalismus des Chinesischen Restesatzes haben wir:

$$m_1 = 3, m_2 = 7, m_3 = 11.$$

Dann ist

$$m = \prod_{i=1}^{3} m_i = m_1 \cdot m_2 \cdot m_3 = 231.$$

Dann ist

$$\xi_1 = m_2 \cdot m_3 = 77$$
$$\xi_2 = m_1 \cdot m_3 = 33$$
$$\xi_3 = m_1 \cdot m_2 = 21.$$

Damit ist:

$$\xi_1^{-1} \bmod m_1 \equiv 77^{-1} \bmod 3 \equiv 2^{-1} \bmod 3 \equiv 2 \bmod 3$$
$$\xi_2^{-1} \bmod m_2 \equiv 33^{-1} \bmod 7 \equiv 5^{-1} \bmod 7 \equiv 3 \bmod 7$$
$$\xi_3^{-1} \bmod m_3 \equiv 21^{-1} \bmod 11 \equiv 10^{-1} \bmod 11 \equiv 10 \bmod 11.$$

Daraus erhalten wir die c_i gemäß:

$$c_1 = \xi_1 \cdot (\xi_1^{-1} \bmod m_1) = 77 \cdot 2 = 154$$
$$c_2 = \xi_2 \cdot (\xi_2^{-1} \bmod m_2) = 33 \cdot 3 = 99$$
$$c_3 = \xi_3 \cdot (\xi_3^{-1} \bmod m_3) = 21 \cdot 10 = 210.$$

Die Lösung a ergibt sich dann zu:

$$\begin{aligned} a &= (a_1c_1 + a_2c_2 + a_3c_3) \bmod m_1m_2m_3 \\ &= (154 + 3 \cdot 99 + 5 \cdot 210) \bmod 231 \\ &= 115. \end{aligned}$$

Check:

$$115 \bmod 3 \equiv 1 \bmod 3$$
$$115 \bmod 7 \equiv 3 \bmod 7$$
$$115 \bmod 11 \equiv 5 \bmod 11.$$

15.5 Endliche Körper

Im Abschnitt [15.2] haben wir auf den Restklassenmengen $\mathbb{Z}_n$ die beiden binären Operationen Addition modulo n und Multiplikation modulo n betrachtet und durch explizite Konstruktion gesehen, daß auf $\mathbb{Z}_5$ für jedes $a \in \mathbb{Z}_5$ ein multiplikatives Inverses in $\mathbb{Z}_5$ existiert, was auf $\mathbb{Z}_6$ nicht der Fall war. In diesem Abschnitt werden wir die algebraischen Strukturen der Restklassen näher untersuchen.

Theorem:

Die folgende Eigenschaft gilt in der Restklassenmenge $\mathbb{Z}_n$ genau dann, wenn n eine Primzahl ist:

$$\text{Wenn } a \cdot b = 0 \text{ dann folgt } a = 0 \text{ oder } b = 0$$

Beweis:

Sei $n = p$ eine Primzahl und es gilt

$$a \cdot b = 0 \text{ auf der Menge } \mathbb{Z}_p$$

Dies ist äquivalent zur Kongruenz

$$a \cdot b \equiv 0 \bmod p$$

Dies ist dann der Fall, wenn die Zahl p Teiler des Produktes $a \cdot b$ ist. Da p eine Primzahl ist, teilt diese Zahl das Produkt genau dann, wenn sie wenigstens einen der Faktoren teilt. Ist p Teiler des Faktors a, dann ist

$$a \equiv 0 \bmod p$$

i.e. $a = 0$ in der Menge $\mathbb{Z}_p$. Analog argumentiert man, wenn p Teiler von b ist, dann ist

$$b \equiv 0 \bmod p$$

und damit $b = 0$ in $\mathbb{Z}_p$.

Die Primzahleigenschaft von p ist die entscheidende Eigenschaft der obigen Argumentation. Falls n keine Primzahl ist, muß n die Form haben:

$$n = \alpha \cdot \beta, \text{ mit } 2 \leq \alpha, \beta < n - 1$$

In that case, α and β are nonzero elements of $\mathbb{Z}_n$, with n divides the product $\alpha \cdot \beta$ without remainder. That is:

$$\alpha \cdot \beta \equiv 0 \,(\bmod n)$$

that implies: $\alpha \cdot \beta = 0$ in $\mathbb{Z}_n$

Satz:

Die Menge $\mathbb{Z}_n$ der ganzen Zahlen modulo n mit den beiden Operationen Addition modulo n und Multiplikation modulo n ist ein Körper genau dann, wenn n eine Primzahl ist.

Beweis:

Suppose first that $n = p$ is a prime number. We will not establish all the properties in the definition of a field, but show only that every nonzero element in $\mathbb{Z}_p$ has an inverse. Consider all the products of a with the elements of $\mathbb{Z}_p$:

$$\{a \cdot b \mid b \in \mathbb{Z}_p\}$$

These p products are distinct, for if $a \cdot b = a \cdot b'$, we get $a \cdot (b - b') = 0$ and Theorem 1 implies $b - b' = 0$, hence $b = b'$. Since the p products are distinct, they represent every element in $\mathbb{Z}_p$. In particular, on of these products must equal 1, that is $a \cdot b = 1$ for some $b_1 \in \mathbb{Z}_p$. By commutativity we also have $b_1 \cdot a = 1$, and so b_1 is the invers of a.

If n is not a prime then $n = \alpha \cdot \beta$ for some nonzero $\alpha, \beta \in \mathbb{Z}_n$. As in the proof of Theorem 1, it follows that $\alpha \cdot \beta = 0$ in $\mathbb{Z}_n$. and therefore, α cannot have an inverse. For if α^{-1} did exist, then we would have:

$$\beta = \alpha^{-1}(\alpha \cdot \beta) = \alpha^{-1} \cdot 0 = 0$$

which is not the case. Since the nonzero element α does not have an inverse, the set $\mathbb{Z}_n$ cannot be field.

Examples:

Since $2 \cdot 4 = 1$ in $\mathbb{Z}_7$, the inverse of 2 is 4, that is $2^{-1} = 4$. Recall, this is true only in the field $\mathbb{Z}_7$ and not in the field of real numbers.

When p is not a prime, all of the properties in the definition of a field hold except that not all nonzero elements have inverses. However, since $\mathbb{Z}_n$ is not a field when n is not prime, many properties that hold in $\mathbb{Z}_p$ do not hold in $\mathbb{Z}_n$.

The field $\mathbb{Z}_2$ of integers modulo 2 has an interesting property not shared by the other fields $\mathbb{Z}_p$, $p > 2$. Since $1 + 1 = 0$ in $\mathbb{Z}_2$, we have $1 = -1$. Certainly, $0 = -0$ and so, if e is either 0 or 1, then:

$$e - e = e + (-e) = e + e$$

In other words, in $\mathbb{Z}_2$ each element is its own negative and subtraction is the same as addition. Note that this is true if and only if $p = 2$.

Definition:

A field of order q is called a **Galois field** and is denoted by $GF(q)$.

The fields with a finite number of elements – for instance the set $\mathbb{Z}_2$ together with the operations addition and multiplication – were studied by the French mathematician **Evariste Galois** (1811-1832) who died in a duel at the age of 20.

Kapitel 16

Gruppen

16.1 Überblick

Die algebraische Struktur der Gruppe haben wir bereits in Kapitel [13.2] kennengelernt, hier zur Erinnerung nochmals die Definition:

Definition:

Eine **Gruppe** ist eine Halbgruppe $(\mathbb{A}, \circ)$ mit den folgenden Eigenschaften:

1. Es gibt ein *neutrales Element* $\mathbf{1}_{\mathbb{A}} \in \mathbb{A}$ mit

$$a \circ \mathbf{1}_{\mathbb{A}} = a \qquad \forall\, a \in \mathbb{A}$$

2. Für jedes $a \in \mathbb{A}$ existiert ein Element $a^{-1} \in \mathbb{A}$ – das *Inverse* von a – mit

$$a \circ a^{-1} = \mathbf{1}_{\mathbb{A}} \qquad \forall a \in \mathbb{A}$$

Behauptung:

Sei $G = (\mathbb{A}, \circ)$ eine Gruppe. Seien $a, b \in G$. Dann sind die Gleichungen

$$a \circ x = b \qquad \text{und} \qquad y \circ a = b \tag{16.1}$$

in G eindeutig lösbar, i.e. es gibt Elemente $x, y \in \mathbb{A}$, so dass die Gleichungen (16.1) in der Menge $\mathbb{A}$ lösbar sind.

Beweis:

Die Gleichung $a \circ x = b$ ist in G lösbar. Zu $a \in G$ existiert – da G per annahme eine Gruppe ist – ein $a^{-1} \in G$. Setzen wir

$$x = a^{-1} \circ b,$$

dann folgt:

$$\begin{aligned} b &= a \circ x \\ &= a \circ (a^{-1} \circ b) \\ &= (a \circ a^{-1}) \circ b \qquad \text{Assoziativität von } \circ \\ &= \mathbf{1}_G \circ b \qquad \text{Existenz des neutralen Elements} \\ &= b. \end{aligned}$$

Die Gleichung ist eindeutig lösbar: Angenommen, es gibt zwei Elemente $x, x' \in G$, die die Gleichung $a \circ x = b$ lösen. Dann ist

$$a \circ x = b \quad \text{und } a \circ x' = b.$$

Damit haben wir

$$a \circ x = a \circ x'$$

oder:

$$(a^{-1} \circ a) \circ x = (a^{-1} \circ a) \circ x'.$$

Hieraus folgt $x = x'$, i.e. die *Eindeutigkeit* der Lösung.

Auf analoge Weise zeigt man die Existenz sowie die Eindeutigkeit der Lösung der Gleichung $y \circ a = b$.

Definition:

Eine Gruppe heißt *endlich*, falls sie endlich viele Elemente hat. Die Anzahl der Elemente einer endlichen Gruppe nennt man die **Ordnung der Gruppe**.

Wie wir bereits in Kapitel [15] ausführlich diskutiert haben, definiert die Kongruenzrelation $a \equiv b \bmod n$ eine Äquivalenzrelation auf der Menge $\mathbb{Z}$ der ganzen Zahlen. Die **Äquivalenzklassen** dieser Äquivalenzrelation sind die Restklassen:

$$\begin{aligned} [0] &= \{\dots, -2n, -n, 0, n, 2n, \dots\} \\ [1] &= \{\dots, -2n+1, -n+1, 1, n+1, 2n+1, \dots\} \\ &\vdots \quad \vdots \\ [n-1] &= \{\dots, -n-1, -1, n-1, 2n-1, 3n-1, \dots\}. \end{aligned}$$

Auf der Menge der Restklassen modulo n können wir eine binäre Operation definieren (siehe auch Gl. (15.1)):

$$[a] \oplus [b] = [a+b] \tag{16.2}$$

wobei a und b irgendwelche Elemente der Äquivalenzklassen $[a]$ respektive $[b]$ sind (Repräsentanten).

Um zu zeigen, dass dadurch tatsächlich eine Operation definiert wird – i.e. dass diese Operation wohldefiniert ist – muß man zeigen, dass die Operation (16.2) unabhängig von den Repräsentanten a bzw. b ist. Seien also $a' \in [a]$ und $b' \in [b]$ zwei beliebige Repräsentanten der Restklassen $[a]$ bzw. $[b]$. Zu zeigen ist, dass

$$[a' + b'] = [a + b].$$

Da $a' \in [a]$ und $b' \in [b]$ folgt, es gibt Zahlen $s, t \in \mathbb{Z}$ mit

$$a' = a + s \cdot n \text{ und } b' = b + t \cdot n.$$

Damit ist aber:

$$a' + b' = a + s \cdot n + b + t \cdot n = a + b + (s + t) \cdot n \equiv a + b \bmod n.$$

Daher ist die Operation (16.2) wohldefiniert, i.e. unabhängig von den Repräsentanten der Restklasse. Die Assoziativität der $\oplus$-Operation folgt aus der Assoziativität der gewöhnlichen Addition, die Restklasse $[0]$ bildet das neutrale Element und $[-a]$ ist das Inverse von $[a]$. Daher bilden die Elemente der Menge

$$\Big\{[0], [1], \ldots, [n-1]\Big\}$$

eine Gruppe.

Definition

Die Gruppe, die durch die Menge

$$\Big\{[0], [1], \ldots, [n-1]\Big\}$$

der Restklassen modulo n mit der Operation (16.2) gebildet wird, heißt **Gruppe der ganzen Zahlen modulo** n und wird mit $\mathbb{Z}_n$ bezeichnet.

Die Gruppe $(\mathbb{Z}_n, \oplus)$ ist eine zyklische Gruppe mit der Restklasse $[1]$ als Erzeuger und der Ordnung n.

Es gibt eine bequeme Weise, eine endliche Gruppe darzustellen. Mit Hilfe einer Tafel wird die Gruppenoperation dargestellt, heute nennt man diese Tafeln auch **Cayley Tafeln**. Im folgenden ist die CAYLEY Tafel der Gruppe $(\mathbb{Z}_6, \oplus)$ aufgeführt.

$\oplus$	$[0]$	$[1]$	$[2]$	$[3]$	$[4]$	$[5]$
$[0]$	$[0]$	$[1]$	$[2]$	$[3]$	$[4]$	$[5]$
$[1]$	$[1]$	$[2]$	$[3]$	$[4]$	$[5]$	$[0]$
$[2]$	$[2]$	$[3]$	$[4]$	$[5]$	$[0]$	$[1]$
$[3]$	$[3]$	$[4]$	$[5]$	$[0]$	$[1]$	$[2]$
$[4]$	$[4]$	$[5]$	$[0]$	$[1]$	$[2]$	$[3]$
$[5]$	$[5]$	$[0]$	$[1]$	$[2]$	$[3]$	$[4]$

Eine Gruppe G enthält bestimmte Teilmengen, die selbst wieder die Eigenschaft einer Gruppe bezüglich der Operation auf G hat. Man erkennt leicht, dass die Teilmenge

$$\{[0],[2],[4]\} \subset \mathbb{Z}_6$$

diese Eigenschaft hat. Diese Beobachtung führt auf das Konzept der **Untergruppe**. Dies wird in Abschnitt (16.4) näher betrachtet.

Beispiel:

Das folgende Beispiel soll die in diesem Kapitel untersuchten Aspekte über Gruppen weiter motivieren.

Wir betrachten die Menge

$$\mathbb{Z}_{15} = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$$

Wir wir in Kapitel [15.2] gesehen haben, kann man auf solchen endlichen Zahlenmengen die modulare Addition und Multiplikation betrachten. Von Interesse ist hier die modulare Multiplikation. Die Menge $\mathbb{Z}_{15}$ ist unter der Multiplikation abgeschlossen, dies ist in der Tabelle [16.1] zu sehen, in der die Multiplikation modulo 15 komplett aufgeführt ist.

$\times$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	0	2	4	6	8	10	12	14	1	3	5	7	9	11	13
3	0	3	6	9	12	0	3	6	9	12	0	3	6	9	12
4	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11
5	0	5	10	0	5	10	0	5	10	0	5	10	0	5	10
6	0	6	12	3	9	0	6	12	3	9	0	6	12	3	9
7	0	7	14	6	13	5	12	4	11	3	10	2	9	1	8
8	0	8	1	9	2	10	3	11	4	12	5	13	6	14	7
9	0	9	3	12	6	0	9	3	12	6	0	9	3	12	6
10	0	10	5	0	10	5	0	10	5	0	10	5	0	10	5
11	0	11	7	3	14	10	6	2	13	9	5	1	12	8	4
12	0	12	9	6	3	0	12	9	6	3	0	12	9	6	3
13	0	13	11	9	7	5	3	1	14	12	10	8	6	4	2
14	0	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Tabelle 16.1: Modulare Multiplikation auf der Menge $\mathbb{Z}_{15}$.

Weiterhin liest man aus der Tabelle ab[1], dass die Zahl 1 das neutrale Element der Multiplikation ist. Ausserdem erkennt man an der Resultaten der Tabelle [16.1], dass es eine Reihe von Elementen $a \neq 0, b \neq 0$ aus $\mathbb{Z}_{15}$ gibt mit der Eigenschaft

$$a \times b \equiv 0 \bmod 15$$

[1] Dies erkennt man aus der zweiten Zeile bzw. zweiten Spalte der Tabelle [16.1].

Präziser sind dies die Elemente $a = 3, 5, 6, 9, 10$ und $a = 12$. Diese Elemente nennt man **Nullteiler**.

Die restlichen Elemente – ausgenommen natürlich die 0 – haben die Eigenschaft, dass sie ein Inverses bezüglich der Multiplikation haben:

$$\begin{aligned} 1 \times 1 &\equiv 1 \bmod 15 \\ 2 \times 8 &\equiv 1 \bmod 15 \\ 4 \times 4 &\equiv 1 \bmod 15 \\ 7 \times 13 &\equiv 1 \bmod 15 \\ 11 \times 11 &\equiv 1 \bmod 15 \\ 14 \times 14 &\equiv 1 \bmod 15 \end{aligned}$$

Generell nennt man die Elemente einer Menge mit Einselement, die bezüglich der Multiplikation invertierbar sind, **Einheiten einer Menge**. Die Einheiten von $\mathbb{Z}_{15}$, i.e. die acht Zahlen 1, 2, 4, 7, 8, 11, 13 und 14 fassen wir in der Menge

$$\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

zusammen.

Wir zeigen zunächst, daß die Menge $\mathbb{Z}_{15}^*$ abgeschlossen ist unter der modularen Multiplikation. Dazu schreiben wir die Multiplikationstafel dieser Menge modulo 15 auf, diese ist in der Tabelle [16.2] aufgelistet.

$\times$	**1**	**2**	**4**	**7**	**8**	**11**	**13**	**14**
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

Tabelle 16.2: Multiplikation auf der Menge $\mathbb{Z}_{15}^*$.

Das Paar $(\mathbb{Z}_{15}^*, \times)$ bildet eine Gruppe, da

1. $\mathbb{Z}_{15}^*$ abgeschlossen ist unter $\times$
2. das Assoziativgesetz gilt
3. $1 \in \mathbb{Z}_{15}^*$
4. und für alle $a \in \mathbb{Z}_{15}^*$ ist auch $a^{-1} \in \mathbb{Z}_{15}^*$.

Diese Gruppe nennt man **Einheitengruppe** (von $\mathbb{Z}_{15}$).

Beispiel:

Eine weitere Menge, die sowohl in der Theorie der endlichen Gruppen als auch in der Kryptographie eine wichtige Rolle spielt sind die Menge der **Permutationen**.

Definition:

Eine Permutation einer Menge $A = \{a_1, a_2, \ldots, a_n\}$ ist eine bijektive Abbildung

$$\pi \; : \; A \longrightarrow A$$

d.h. jedem Element a der Menge A wird ein Element $\pi(a)$ zugeordnet, jedes Element von A ist das Bild genau eines a.

Notation:

Eine Permutation π einer Menge $A = \{a_1, a_2, \ldots, a_n\}$ schreibt man in der Form:

$$\pi = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ \pi(a_1) & \pi(a_2) & \cdots & \pi(a_n) \end{pmatrix}$$

Es gibt $n!$ verschiedene Permutationen einer Menge mit n Elementen.

Die folgende Tabelle ist eine Liste sämtlicher bijektiven Abbildungen σ_i der Menge $\{1, 2, 3\}$ in sich selbst auf:

$$\sigma_i \; : \; \{1, 2, 3\} \longrightarrow \{1, 2, 3\}$$

	1	2	3
σ_0	1	2	3
σ_1	2	3	1
σ_2	3	1	2
σ_3	1	3	2
σ_4	3	2	1
σ_5	2	1	3

Theorem

Die algebraische Struktur

$$\begin{aligned} \mathcal{S}_n &= \Big(\{\sigma \; : \; \{1, \ldots, n\} \longrightarrow \{1, \ldots n\} \mid \sigma \text{ bijektiv } \}, \circ\Big) \\ &= \Big(\{\sigma_0, \sigma_1, \ldots, \sigma_{n!-1}\}, \circ\Big), \end{aligned}$$

wobei die Abbildungen $\sigma_i, i = 0, 1, \ldots, n! - 1$ die Permutationen der Menge $\{1, 2, \ldots, n\}$ sind und die binäre Operation $\circ$ ist die Hintereinanderausführung, ist eine Gruppe der Ordnung $n!$ mit dem neutralen Element σ_0.

Definition:

Die Gruppe $\mathcal{S}_n$ heißt die **vollständige symmetrische Gruppe** der Ordnung n.

Beispiel:

Die obige Tabelle enthält alle Elemente der Gruppe $\mathcal{S}_3$. Die folgende Tabelle zeigt die Verknüpfung dieser Gruppe:

$\circ$	σ_0	σ_1	σ_2	σ_3	σ_4	σ_5
σ_0	σ_0	σ_1	σ_2	σ_3	σ_4	σ_5
σ_1	σ_1	σ_2	σ_0	σ_4	σ_5	σ_3
σ_2	σ_2	σ_0	σ_1	σ_5	σ_3	σ_4
σ_3	σ_3	σ_5	σ_4	σ_0	σ_2	σ_1
σ_4	σ_4	σ_3	σ_5	σ_1	σ_0	σ_2
σ_5	σ_5	σ_4	σ_5	σ_2	σ_1	σ_0

Die Einträge dieser Tabelle erhält man durch folgende Rechnungen:

$$\begin{aligned}(\sigma_1 \circ \sigma_1)(1,2,3) &= \sigma_1(\sigma_1(1,2,3)) \\ &= \sigma_1(2,3,1) \\ &= (3,1,2) \\ &= \sigma_2(1,2,3).\end{aligned}$$

$$\begin{aligned}(\sigma_1 \circ \sigma_2)(1,2,3) &= \sigma_1(\sigma_2(1,2,3)) \\ &= \sigma_1(3,1,2) \\ &= (1,2,3) \\ &= \sigma_0(1,2,3)\end{aligned}$$

Da σ_0 das neutrale Element der Gruppe $\mathcal{S}_3$ ist, folgt:

$$\sigma_1^{-1} = \sigma_2.$$

$$\begin{aligned}(\sigma_1 \circ \sigma_3)(1,2,3) &= \sigma_1(\sigma_3(1,2,3)) \\ &= \sigma_1(1,3,2) \\ &= (3,2,1) \\ &= \sigma_4(1,2,3)\end{aligned}$$

$$\begin{aligned}(\sigma_1 \circ \sigma_4)(1,2,3) &= \sigma_1(\sigma_4(1,2,3)) \\ &= \sigma_1(3,2,1) \\ &= (2,1,3) \\ &= \sigma_5(1,2,3)\end{aligned}$$

$$\begin{aligned}(\sigma_1 \circ \sigma_5)(1,2,3) &= \sigma_1(\sigma_5(1,2,3)) \\ &= \sigma_1(2,1,3) \\ &= (1,3,2) \\ &= \sigma_3(1,2,3).\end{aligned}$$

usw.

Definition:

Jede Untergruppe einer vollständigen symmetrischen Gruppe heißt **Permutationsgruppe**.

Beispiele:

Wie man aus der obigen Verknüpfungstabelle der symmetrischen Gruppe $\mathcal{S}_3$ abliest, sind

- ✗ $(\{\sigma_0\}, \circ)$
- ✗ $(\{\sigma_0, \sigma_1, \sigma_2\}, \circ)$
- ✗ $(\{\sigma_0, \sigma_5\}, \circ)$

sind Permutationsgruppen.

Der folgende Satz von Arthur Cayley sagt aus, dass die endlichen Gruppen im wesentlichen durch Permutationsgruppen beschrieben werden können[2].

Theorem (Satz von Cayley)

Jede endliche Gruppe ist isomorph zu einer Permutationsgruppe.

Beweis:

Sei $G = (\mathbb{A}, \circ)$ eine endliche Gruppe der Ordnung n, das neutrale Element dieser Gruppe bezeichnen wir mit $\mathbf{1}_G \in \mathbb{A}$. Wir definieren für jedes $a \in \mathbb{A}$ eine Abbildung:

$$f_a : \mathbb{A} \longrightarrow \mathbb{A}$$

durch

$$f_a(x) = a \circ x$$

Gemäß den Gleichungen (16.1) kann *jedes* $g \in G$ in der Form $g = a \circ x$ geschrieben werden, daher ist die Abbildung

$$f_a : \mathbb{A} \longrightarrow \mathbb{A}$$

surjektiv. Diese Abbildung ist auch injektiv, da die Implikation

$$f_a(x) = f_a(y) \implies x = y$$

gilt. Denn:

$$f_a(x) = f_a(y) \implies a \circ x = a \circ y.$$

[2] Siehe auch [114, S. 29 ff.], die symmetrischen Gruppen heißen dort Transformationsgruppen.

Verknüpft man diese Gleichung von links mit dem Element $a^{-1} \in G$, erhält man:

$$a^{-1} \circ (a \circ x) = a^{-1} \circ (a \circ y).$$

Da das Assoziativgesetz gilt, folgt die Identität:

$$(a^{-1} \circ a) \circ x = (a^{-1}\circ)a \circ y.$$

Da

$$a^{-1} \circ a = \mathbf{1}_G$$

erhalten wir:

$$\mathbf{1}_G \circ x = \mathbf{1}_G \circ y.$$

oder

$$x = y.$$

Damit ist gezeigt, dass die Abbildung

$$f_a \; : \; \mathbb{A} \longrightarrow \mathbb{A}$$

eine Bijektion ist.
Die Struktur

$$F^M = \big(\{f_a \mid a \in \mathbb{A}\}, \circ\big)$$

bildet eine Gruppe. Dazu müssen wir zeigen:

1. Assoziativität:

$$\begin{aligned}\Big(f_a \circ (f_b \circ f_c)\Big)(x) &= f_a \circ (f_b \circ f_c(x)) \\ &= f_a \circ (f_b(c \circ x)) \\ &= f_a \circ \big(b \circ (c \circ x)\big) \\ &= a \circ \big(b \circ (c \circ x)\big) \\ &= \big(a \circ b\big) \circ c \circ x \\ &= \big((f_a \circ f_b) \circ f_c\big)(x)\end{aligned}$$

16.2 Gruppenhomorphismen

Vergleicht man die Strukturen zweier Gruppen, dann spielen solche Abbildungen zwischen diesen Gruppen eine wichtige Rolle, die die Operationen erhalten.

Definition

Seien $(G_1, \times)$ und $(G_2, \otimes)$ zwei Gruppen. Eine Abbildung

$$f : G_1 \longrightarrow G_2$$

der Gruppe G_1 in die Gruppe G_2 heißt ein **Homomorphismus** von G_1 in G_2, wenn f die Operation auf G_1 erhält, i.e.

$$f(a \times b) = f(a) \otimes f(b) \quad \forall\, a, b \in G_1.$$

Falls zusätzlich f surjektiv ist, dann heißt f ein **Epimorphismus**. Ein Homomorphismus von G_1 auf G_1 nennt man **Endomorphismus**. Ist f eine bijektive Abbildung der Gruppe G_1 in die Gruppe G_2, dann ist f ein **Isomorphismus**. Ein Isomorphismus von G_1 auf G_1 heißt **Automorphismus**.

Folgerungen:

Sei ρ ein Gruppenisomorphismus

$$\rho : G_1 \longrightarrow G_2$$

mit den Gruppen $(G_1, \times)$ bzw. $(G_2, \otimes)$. Weiterhin ist 1_{G_1} Identität der Gruppe G_1 und 1_{G_2} Einselement der Gruppe G_2. Dann gilt

❶ $\rho(1_{G_1}) = 1_{G_2}$

❷ $\rho(a^{-1}) = (\rho(a))^{-1}$

Beweis:

Wir setzen

$$\rho(a_1) = a_2$$

Dann gilt:

$$\begin{aligned} a_2 \otimes 1_{G_2} &= a_2 \\ &= \rho(a_1) \\ &= \rho(a_1 \times 1_{G_1}) \\ &= \rho(a_1) \otimes \rho(1_{G_1}) \\ &= a_2 \otimes \rho(1_{G_1}) \end{aligned}$$

Da in Gruppen die Kürzungsregel gilt, folgt die Behauptung.

Um die zweite Folgerung zu zeigen, bemerken wir, daß das Einselement der Gruppe $(G_2, \otimes)$ geschrieben werden kann in der Form:

$$1_{G_2} = \rho(a) \otimes (\rho(a))^{-1}$$

Andererseits ist nach dem zuvor gezeigten:

$$\begin{aligned} 1_{G_2} &= \rho(1_{G_1}) \\ &= \rho(a \times a^{-1}) \\ &= \rho(a) \otimes \rho(a^{-1}) \end{aligned}$$

Daher folgt:

$$\rho(a) \otimes (\rho(a))^{-1} = \rho(a) \otimes \rho(a^{-1})$$

Hieraus folgt wieder anhand der Kürzungsregel die Behauptung

$$\rho(a^{-1}) = (\rho(a))^{-1}$$

Beispiele:

① Sei $G = (\mathbb{Z}, +)$ die additive Gruppe der ganzen Zahlen und sei $H = (\mathbb{Z}_n, \oplus)$ die additive Gruppe der ganzen Zahlen modulo n. Dann ist

$$f :: G \longrightarrow H$$

mit

$$f(a) = [a]$$

ein Gruppenhomomorphismus, da

$$f(a+b) = [a+b] = [a] + [b] = f(a) + f(b).$$

② Betrachte die beiden Abelschen Gruppen

$$G_1 = (\mathbb{R}_+, \times) \text{ und } G_2 = (\mathbb{R}, +)$$

G_1 ist die multiplikative Gruppe der positiven reellen Zahlen G_1. Das neutrale Element dieser Gruppe ist die 1, da

$$1 \times a = a \times 1 = a \quad \forall\, a \in \mathbb{R}_+$$

und das inverse Element ist die Zahl $a^{-1} \in \mathbb{R}_+$ mit:

$$a \times a^{-1} = a^{-1} \times a = 1 \quad \forall\, a \in \mathbb{R}_+$$

Die Gruppe G_2 ist die additive Gruppe der reellen Zahlen mit dem neutralen Element 0, da

$$0 + a = a + 0 = a \quad \forall\, a \in \mathbb{R}$$

und das additive Inverse ist die Zahl $(-a) \in \mathbb{R}$ mit

$$a + (-a) = (-a) + a = 0 \quad \forall\, a \in \mathbb{R}$$

Betrachte nun den Logarithmus als Abbildung

$$\texttt{log} \ : \ G_1 \longrightarrow G_2$$

Der Logarithmus ist ein Gruppenisomorphismus, denn es gilt:

$$\texttt{log}(a \times b) = \texttt{log}(a) + \texttt{log}(b)$$

Weiterhin gelten die Rechenregeln:

$$\texttt{log}(1) = 0$$

und

$$\texttt{log}(a^{-1}) = -\texttt{log}(a)$$

Die Umkehrabbildung des Logarithmus – die Exponentialfunktion – ist ebenfalls ein Gruppenisomorphismus.

③ Betrachte die Gruppen

$$G_1 = (\mathbb{Z}_4, +)$$

und

$$G_2 = (\mathbb{Z}_5^*, \times)$$

Diese beiden Gruppen können durch Verknüpfungstafeln dargestellt werden.

$+$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Tabelle 16.3: Gruppentafel von $(\mathbb{Z}_4, +)$.

Wie die Gruppentafel [16.3] zeigt, ist das Einselement dieser Gruppe die 0, das Inverse zu $a \in \mathbb{Z}_4$ ist das Element

$$a^{-1} = (4 - a) \bmod 4$$

$\mathbb{Z}_4$ ist damit eine ABELsche Gruppe mit vier Elementen.

Die Menge $\mathbb{Z}_5^*$ ist ebenfalls eine ABELsche Gruppe mit Einselement 1. Die Elemente 1 und 4 sind zu sich selbst invers, 2 und 3 sind invers zueinander.

$\times$	**1**	**2**	**3**	**4**
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Tabelle 16.4: Gruppentafel von $(\mathbb{Z}_5^*, \times)$.

Die dritte Gruppe, die wir betrachten, konstruieren wir folgendermaßen. Sei $\mathbb{Z}_2 = \{0, 1\}$ und betrachte das kartesische Produkt, i.e. die Menge der geordneten Paare:

$$\mathbb{Z}_2 \times \mathbb{Z}_2 = \big\{(0,0), (0,1), (1,0), (1,1)\big\}$$

Auf dieser Menge definieren wir nun eine binäre Abbildung:

$$\begin{aligned} \texttt{xor} \ : \Big(\mathbb{Z}_2 \times \mathbb{Z}_2\Big) \times \Big(\mathbb{Z}_2 \times \mathbb{Z}_2\Big) &\longrightarrow \mathbb{Z}_2 \times \mathbb{Z}_2 \\ \big((a,b),(c,d)\big) &\longmapsto (a,b) \ \texttt{xor} \ (c,d) \stackrel{\text{def}}{=} (a \ \texttt{xor} \ c, b \ \texttt{xor} \ d) \end{aligned}$$

Die `xor` Operation operiert also gemäß dieser Definition komponentenweise. Die Verknüpfungstafel ist in der Tabelle [16.5] dargestellt.

`xor`	`(0,0)`	`(0,1)`	`(1,0)`	`(1,1)`
`(0,0)`	`(0,0)`	`(0,1)`	`(1,0)`	`(1,1)`
`(0,1)`	`(0,1)`	`(0,0)`	`(1,1)`	`(1,0)`
`(1,0)`	`(1,0)`	`(1,1)`	`(0,0)`	`(0,1)`
`(1,1)`	`(1,1)`	`(1,0)`	`(0,1)`	`(0,0)`

Tabelle 16.5: Verknüpfungstafel von $(\mathbb{Z}_2 \times \mathbb{Z}_2, \texttt{xor}\ )$.

Aus dieser Tabelle liest man folgende Eigenschaften ab:

① Die Menge $\mathbb{Z}_2 \times \mathbb{Z}_2$ ist abgeschlossen unter der `xor` Operation.

② Das Element `(0,0)` ist das neutrale Element der `xor` Verknüpfung auf $\mathbb{Z}_2 \times \mathbb{Z}_2$, denn

$$(0,0) \ \texttt{xor} \ (\texttt{a},\texttt{b}) = (\texttt{a},\texttt{b}) \ \texttt{xor} \ (0,0) = (\ \texttt{a},\texttt{b}\ )$$

für alle $(\ \texttt{a},\texttt{b}) \in \mathbb{Z}_2 \times \mathbb{Z}_2$.

③ Jedes Element ist zu sich selbst invers, i.e.

$$(\texttt{a},\texttt{b}) \ \texttt{xor} \ (\texttt{a},\texttt{b}) = (0,0)$$

Diese EIgenschaft nennt man **Idempotenz**.

④ Die Assoziativität dieser Verknüpfung ist leicht nachzuvollziehen.

⑤ Die Verknüpfung ist kommutativ.

Damit ist $K_4 = (\mathbb{Z}_2 \times \mathbb{Z}_2, \texttt{xor})$ eine ABELsche Gruppe der Ordnung 4.

Die Frage ist, welche der Gruppen G_1, G_2, K_4 nun zueinander isomorph sind.

Betrachte die Abbildung

$$\xi \; : \; G_1 \longrightarrow G_2$$

i.e.

$$\xi \; : \; \mathbb{Z}_4 \longrightarrow \mathbb{Z}_5^*$$

mit:

$$\begin{array}{ll} \xi(0) = 1 & \xi(1) = 2 \\ \xi(2) = 4 & \xi(3) = 3 \end{array}$$

Dann gilt:

$$\begin{aligned}
\xi(0+0) = \xi(0) &\stackrel{\text{def}}{=} 1 = 1 \times 1 \stackrel{\text{def}}{=} \xi(0) \times \xi(0) \\
\xi(0+1) = \xi(1) &\stackrel{\text{def}}{=} 2 = 1 \times 2 \stackrel{\text{def}}{=} \xi(0) \times \xi(1) \\
\xi(0+2) = \xi(2) &\stackrel{\text{def}}{=} 4 = 1 \times 4 \stackrel{\text{def}}{=} \xi(0) \times \xi(2) \\
\xi(0+3) = \xi(3) &\stackrel{\text{def}}{=} 3 = 1 \times 3 \stackrel{\text{def}}{=} \xi(0) \times \xi(3) \\
\xi(1+1) = \xi(2) &\stackrel{\text{def}}{=} 4 = 2 \times 2 \stackrel{\text{def}}{=} \xi(1) \times \xi(1) \\
\xi(1+2) = \xi(3) &\stackrel{\text{def}}{=} 3 = 2 \times 4 \stackrel{\text{def}}{=} \xi(1) \times \xi(2) \\
\xi(1+3) = \xi(0) &\stackrel{\text{def}}{=} 1 = 2 \times 3 \stackrel{\text{def}}{=} \xi(1) \times \xi(3) \\
\xi(2+2) = \xi(0) &\stackrel{\text{def}}{=} 1 = 4 \times 4 \stackrel{\text{def}}{=} \xi(2) \times \xi(2) \\
\xi(2+3) = \xi(1) &\stackrel{\text{def}}{=} 2 = 4 \times 3 \stackrel{\text{def}}{=} \xi(2) \times \xi(3) \\
\xi(3+3) = \xi(2) &\stackrel{\text{def}}{=} 4 = 3 \times 3 \stackrel{\text{def}}{=} \xi(3) \times \xi(3)
\end{aligned}$$

Mit anderen Worten, die so definierte Abbildung ist verknüpfungstreu und bildet das Einselement der Gruppe G_1 auf das Einselement der Gruppe G_2 ab. Die Abbildung ξ ist daher ein Gruppenisomorphismus. Da wir einen Gruppenisomorphismus angeben können sind die beiden Gruppen G_1 und G_2 bis auf Umbennenung ihrer Elemente gleich.

Die Gruppe $\mathbb{K}_4$ ist nicht isomorph zu G_1. Dies kann folgendermaßen eingesehen werden. Die Gruppentafel [16.5] zeigt, daß jedes Element zu sich selbst invers ist. In der Gruppe $(\mathbb{Z}_4, +)$ sind nur die beiden Elemente 0 und 2 zu sich selbst invers. Aus diesem Grund gibt es keine bijektive Abbildung, die Elemente von $\mathbb{Z}_4$ auf $\mathbb{Z}_2 \times \mathbb{Z}_2$ abbildet, so dass die Verknüpfung erhalten bleibt.

Wir haben in diesem Beispiel daher zwei Gruppen mit jeweils vier Elementen konstruiert, die nicht zueinander isomorph sind. Dieses Ergebnis faßt der folgende Satz (ohne Beweis) zusammen.

Theorem:

Sei $G = (M, \times)$ eine Gruppe mit $\mid M \mid = 4$. Dann ist entweder $G \cong \mathbb{Z}_4$ oder $G \cong \mathbb{K}_4$.

Für die grundlegenden Untersuchungen von algebraischen Eigenschaften vierelementiger Gruppen reicht es daher aus, die beiden Gruppen $\mathbb{Z}_4$ und $\mathbb{K}_4$ zu betrachten. Die Gruppe $\mathbb{K}_4$ heißt nach dem Mathematiker FELIX KLEIN (1849 – 1925) KLEINsche Vierergruppe.

Definition:

Der **Kern** des Homomorphismus

$$f : G \longrightarrow H$$

von der Gruppe G in die Gruppe H ist die Menge

$$\ker f = \{a \in G \mid f(a) = \nVdash_H\},$$

wobei $\nVdash_H$ das Einselement der Gruppe H ist.

Beispiel:

Für den Homomorphismus

$$f : \mathbb{Z} \longrightarrow \mathbb{Z}_n$$

mit

$$f(a) = [a]$$

und

$$f(a + b) = [a + b] = [a] + [b] = f(a) + f(b)$$

besteht der Kern aus allen $a \in \mathbb{Z}$ für die gilt

$$f(a) = [a] = [0].$$

Da dies genau für alle Vielfache a von n gilt. Oder anders ausgedrückt,

$$\ker f =< n >$$

i.e. der Kern von f ist die Untergruppe (siehe Abschnitt [16.4]) von $\mathbb{Z}$, die durch n erzeugt wird.

16.3 Zyklische Gruppen

Sei nun $G = (M, \circ)$ eine Gruppe mit Einselement 1_G. Für jedes Element $a \in M$ führen wir die folgende Notation ein:

$$\begin{aligned} a^0 &= 1_G \\ a^{n+1} &= a^n \circ a \quad n \geq 1 \\ a^{-(n+1)} &= a^{-n} \circ a^{-1} \quad n \geq 1 \end{aligned}$$

Folgerungen:

Aus der Definition der Potenzschreibweise resultieren unmittelbar die folgenden Eigenschaften:

- ✘ $(a^m)^n = a^{m \cdot n} \; \forall \, m, n \in \mathbb{Z}$
- ✘ $a^m \circ a^n = a^{m+n} \; \forall \, m, n \in \mathbb{Z}$
- ✘ $(a^m)^{-1} = (a^{-1})^m = a^{-m}$

Definition:

Sei $G = (M, \circ)$ eine Gruppe. G heißt **zyklisch**, wenn ein Element $a \in M$ existiert, so dass für alle $x \in M$ ein $k \in \mathbb{Z}$ existiert mit $x = a^k$. G wird folgendermaßen notiert: $G = (< a >, \circ)$; das Element a heißt erzeugendes Element von G.

Beispiele:

❶ Wir betrachten die sechselementige Menge

$$\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$$

Man rechnet direkt nach, dass $(\mathbb{Z}_9^*, \times)$ eine Abelsche Gruppe ist. Dies zeigt die Tabelle [16.6].

$\times$	**1**	**2**	**4**	**5**	**7**	**8**
1	1	2	4	5	7	8
2	2	4	8	1	5	7
4	4	8	7	2	1	5
5	5	1	2	7	8	4
7	7	5	1	8	4	2
8	8	7	5	4	2	1

Tabelle 16.6: Gruppentafel von $(\mathbb{Z}_9^*, \times)$.

Nun gilt:

$$\begin{aligned} 2^0 &= 1 \\ 2^1 &= 2 \\ 2^2 &= 3 \\ 2^3 &= 8 \\ 2^4 &= 7 \\ 2^5 &= 5 \end{aligned}$$

Mit anderen Worten, die Gruppe $(\mathbb{Z}_9^*, \times)$ ist zyklisch und das Element $2 \in \mathbb{Z}_9^*$ ist erzeugendes Element der Gruppe $(\mathbb{Z}_9^*, \times)$.

❷ Die Gruppe $(\mathbb{Z}, +)$ ist zyklisch. Das erzeugende Element ist 1. Dabei ist $n = 1^n$ für alle $n \in \mathbb{Z}$ mit

$$1^n = \begin{cases} 1 & \text{falls } n = 0 \\ \sum_{l=1}^{n} 1 & \text{falls } n > 0 \\ \sum_{l=1}^{n} (-1) & \text{falls } n < 0 \end{cases}$$

❸ Die KLEINsche Vierergruppe ist nicht zyklisch, da jedes Element dieser Gruppe zu sich selbst invers ist. Dies wiederum impliziert die Idempotenz, i.e. $a^2 = id$.

Behauptung [232, Chap. 17.2]

1. Sei $G = (M, \circ)$ eine zyklische Gruppe mit erzeugendem Element a. Dann ist G ABELsch.

2. Sei $G = (M, \circ)$ eine zyklische Gruppe mit erzeugendem Element a. Gibt es kein $k \in \mathbb{N}$ mit

$$a^k = \mathbf{1}_G$$

dann ist $a^i \neq a^j$ für alle $i, j \in \mathbb{Z}, i \neq j$ und $M = \{a^i \mid i \in \mathbb{Z}\}$. Die Gruppe G heißt in diesem Fall *unendlich zyklisch*.

3. Sei $G = (M, \times)$ eine zyklische Gruppe mit erzeugendem Element a. Gibt es ein $k \in \mathbb{N}$ mit

$$a^k = e$$

und ist n die kleinste Zahl mit dieser Eigenschaft, dann ist $a^i \neq a^j$ mit $0 \leq i, j \leq n-1, i \neq j$ und:

$$M = \{e, a, a^2, a^3, \ldots, a^{n-1}\}$$

Beweis:

Zu 1: Da G eine zyklische Gruppe mit erzeugendem Element a ist, kann jedes $x \in G$ und jedes $y \in G$ geschrieben werden in der Form:

$$x = a^r \text{ und } y = a^s.$$

Seien also x, y zwei beliebige Elemente der zyklischen Gruppe G mit $r \geq s$ und $r = s + q, q \geq 0$. Dann ist:

$$\begin{aligned} x \circ y &= a^r \circ a^s \\ &= a^{s+q} \circ a^s \\ &= a^{(s+q)+s} \\ &= a^{s+(q+s)} \\ &= a^s \circ a^{s+q} \\ &= a^s \circ a^r \\ &= y \circ x \end{aligned}$$

Damit ist $x \circ y = y \circ x$, daher ist die Gruppe G eine ABELsche Gruppe.

Zu 2: Wir nehmen das Gegenteil der Behauptung an, i.e. es gibt $i, j \in \mathbb{Z}, i \neq j$ mit

$$a^i = a^j.$$

O.B.d.A. sei $j > i$ und $j = j - i$, und damit $k > 0$. Dann ist:

$$\begin{aligned} a^k &= a^{j-i} \\ &= a^j \circ a^{-i} \\ &= a^j \circ (a^i)^{-1} \\ &= a^j \circ (a^j)^{-1} && \text{Annahme } a^i = a^j \\ &= \mathbf{1}_G. \end{aligned}$$

Die ist jedoch ein Widerspruch zur Annahme, dass es kein k gibt mit $a^k = \mathbf{1}_G$.

Da die Gruppe G von dem Element a erzeugt wird und – wie gezeigt – alle Potenzen von a unterschiedlich sind, ist offensichtlich, dass

$$M = \{a^s \mid s \in \mathbb{Z}\}.$$

Zu 3: Annahme: Es gibt i, j mit $0 \leq i, j < n, i \neq j$ so dass $a^i = a^j$. O.B.d.A sei $j > i$ und $k = j - i$, daher ist $k > 0$. Dann ist

$$\begin{aligned} a^k &= a^{j-i} \\ &= a^j \circ a^{-i} \\ &= a^j \circ (a^i)^{-1} \\ &= a^j \circ (a^j)^{-1} && \text{Annahme } a^i = a^j \\ &= \mathbf{1}_G. \end{aligned}$$

Da $k \leq j < n$ ist, haben wir eine Zahl $k < n$ gefunden mit der Eigenschaft $a^k = \mathbf{1}_G$. Dies ist ein Widerspruch dazu, dass n die kleinste Zahl mit dieser Eigenschaft ist.

Jede Zahl $x \in \mathbb{Z}$ kann in der Form

$$x = q \cdot n + r$$

geschrieben werden mit $q \in \mathbb{Z}$ und $0 \leq r < n$. Daher kann ein beliebiges Element der Gruppe G geschrieben werden in der Form:

$$\begin{aligned} a^x &= a^{q \cdot n + r} \\ &= a^{q \cdot n} \circ a^r \\ &= (a^n)^q \circ a^r \\ &= (\mathbf{1}_G)^q \circ a^r \\ &= \mathbf{1}_G \circ a^r \\ &= a^r, \quad 0 \leq r < n. \end{aligned}$$

Da $0 \leq r < n$ folgt hieraus, dass die Menge M aus den Elementen

$$M = \left\{\mathbf{1}_G, a, a^2, \ldots, a^{n-1}\right\}$$

besteht.

Nach diesem Satz gilt also die Aussage, wenn es in einer Gruppe $G = (M, \circ)$ mit Einselement $\mathbf{1}_G$ eine Zahl $n \in \mathbb{N}$ gibt mit $a^n = \mathbf{1}_G$ und ist n die kleinste Zahl mit dieser Eigenschaft, dann ist die Gruppe G endlich, da

$$M = < a > = \left\{\mathbf{1}_G, a, a^2, \ldots, a^{n-1}\right\}.$$

Die Gruppe G hat dann die Ordnung n.

16.4 Untergruppen

Definition:

Sei $G = (M, \circ)$ eine Gruppe. Sei $U \subseteq M$ Teilmenge von M mit:

① Abgeschlossenheit:
Für alle $a, b \in U$ ist auch $a \circ b \in U$.

② Inverse
Für alle $a \in U$ ist auch $a^{-1} \in U$.

Das Paar $G_U = (U, \circ)$ heißt **Untergruppe von G**.

Beispiele:

1. Sei $G = (\mathbb{Z}, +)$ die additive Gruppe der ganzen Zahlen. Die Menge

$$\mathbb{G} = \{\ldots, -4, -2, 0, 2, 4, 6, \ldots\}$$

der geraden Zahlen bildet unter der Additionsoperation eine Untergruppe von G, denn die Menge der geraden Zahlen ist abgeschlossen unter der Additionsoperation. Wenn die Zahl n gerade ist, dann ist auch die Zahl $-n$ gerade und somit sind die Eigenschaften einer Untergruppe erfüllt.

2. Betrachte die KLEINsche Vierergruppe

$$\mathbb{K}_4 = \mathbb{Z}_2 \times \mathbb{Z}_2 = \{(0,0), (0,1), (1,0), (1,1)\}$$

Aus der Gruppentafel [16.5] dieser Gruppe liest man ab, dass

$$U = \{(0,0), (0,1)\}$$

eine Untergruppe ist.

3. Die in Kapitel [13.2] diskutierte CÄSAR Chiffre ist eine Untergruppe der affinen Chiffre.

Behauptung:

Sei $G = (M, \circ)$ eine Gruppe mit Einselement 1_G. Dann gilt:

① G_M und G_{1_G} sind Untergruppen, diese nennt man *triviale* Untergruppen.

② 1_G ist Einselement jeder Untergruppe von G.

③ Die Verknüpfung $\circ$ ist assoziativ in jeder Untergruppe von G. Damit ist jede Untergruppe auch eine Gruppe.

④ Wenn G ABELsche Gruppe ist, dann ist auch jede Untergruppe von G ABELsch.

Beweis:

① Gilt offensichtlich.

② Sei a irgendein Element der Menge $U \subseteq M$ und U ist Untergruppe von G. Da U Untergruppe ist, ist auch $a^{-1} \in U$. Dann muß aber aufgrund der Abgeschlossenheit der Untergruppe auch gelten:

$$a \circ a^{-1} = 1_G \in U$$

Daher muß jede Untergruppe das Einselement 1_G enthalten.

③ Wenn die Verknüpfung $\circ$ in der Untergruppe U nicht assoziativ ist, dann ist sie auch in G nicht assoziativ. Dies ist aber ein Widerspruch zur Gruppeneigenschaft, dass die Verknüpfung einer Gruppe assoziativ ist.

④ Analog führt die Annahme, dass U nicht-kommutativ ist zum Widerspruch, dass die Gruppe G per Annahme ABELsch ist.

Der folgende Satz liefert ein einziges notwendiges und hinreichendes Kriterium für zwei beliebige Elemente einer Teilmenge einer Gruppe dafür, dass diese Teilmenge eine Untergruppe bildet.

Theorem:

Sei $G = (M, \circ)$ eine Gruppe mit Einselement 1_G und $U \subseteq M$. Dann gilt: $G_U = (U, \circ)$ ist Untergruppe von G genau dann, wenn für alle $a, b \in U$ auch $a^{-1} \circ b \in U$ ist.

Beweis:

Es sind zwei Richtungen zu zeigen:

$\Longrightarrow$ Voraussetzung: $G_U = (U, \circ)$ ist Untergruppe von G, $a, b \in U$; zu zeigen, $a^{-1} \circ b \in U$.
Da gemäß Annahme U Untergruppe von G ist, gilt: wenn $a \in U$, dann ist auch $a^{-1} \in U$. Da U auch abgeschlossen unter der $\circ$-Operation ist, ist $a^{-1} \circ b \in U$ für alle $a, b \in U$.

$\Longleftarrow$ Voraussetzung: $a^{-1} \circ b \in U$ für alle $a, b \in U$; zu zeigen ist. U ist Untergruppe von G.
Wenn $a^{-1} \circ b \in U$ gilt, dann können wir auch den Fall $b = a$ betrachten. Dann ist aber $a^{-1} \circ a = 1_G \in U$, i.e. das Einselement von G ist Element der Menge U.

Wenn $a^{-1} \circ b \in U$ gilt und $1_G \in U$, dann können wir auch den Fall $b = 1_G$ betrachten. Dann ist aber $a^{-1} \circ 1_G = a^{-1} \in U$, i.e. für jedes $a \in U$ ist auch $a^{-1} \in U$.

Seien nun $x, y \in U$ zwei beliebige Elemente von U gegeben. Nach dem bisher gezeigten, sind dann die Inverse x^{-1}, y^{-1} ebenfalls Elemente der

Teilmenge U. Setzen wir nun $a = x^{-1}$ und $b = y$, dann muß nach Voraussetzumg auch das Element $a^{-1} \circ b$ Element der Menge U sein, i.e.

$$(x^{-1})^{-1} \circ y = x \circ y \in U$$

Also, damit ist gezeigt, wenn $x, y \in U$, dann ist auch $x \circ y \in U$, i.e. die Menge U ist abgeschlossen unter $\times$.

Damit sind sämtliche Voraussetzungen der Definition einer Untergruppe gezeigt, i.e. U ist Untergruppe von G.

Theorem:

Seien $G_1 = (M_1, \circ_1)$ und $G_2 = (M_2, \circ_2)$ Gruppen und $G_{1U} = (U, \circ_1)$ Untergruppe von G_1. Sei

$$\phi \; : \; M_1 \longrightarrow M_2$$

ein Isomorphismus zwischen den Gruppen G_1 und G_2. Dann ist

$$G_{2\phi(U)} = (\phi(U), \circ_2)$$

eine Untergruppe von G_2.

Beweis:

Unter Zuhilfenahme des vorherigen Satzes, genügt es zu zeigen: wenn aus $a, b \in \phi(U)$ folgt, daß $a^{-1} \circ_2 b \in \phi(U)$, dann ist gemäß vorherigem Satz die Menge $\phi(U)$ mit der Verknüpfung $\circ_2$ Untergruppe von G_2.

Seien also $a, b \in \phi(U)$ gegeben; dann gibt es aber zwei Elemente $x, y \in U$ mit:

$$a = \phi(x) \text{ und } b = \phi(y).$$

Gemäß Voraussetzung ist nun U Untergruppe von G_1, und damit ist nach dem oben gezeigten Satz

$$x^{-1} \circ_1 y \in U.$$

Damit ist aber $\phi(x^{-1} \circ_1 y) \in \phi(U)$ und

$$\begin{aligned} \phi(x^{-1} \circ_1 y) &= \phi(x^{-1}) \circ_2 \phi(y) \\ &= (\phi(x))^{-1} \circ_2 \phi(y) \\ &= a^{-1} \circ_2 b \end{aligned}$$

Daher ist $a^{-1} \circ b \in \phi(U)$.

Definition:

Sei $G = (M, \circ)$ eine Gruppe und $G_U = (U, \circ)$ eine Untergruppe von G. Für jedes $a \in M$ heißt

$$a \circ U = \{a\circ \mid x \in U\}$$

Linksnebenklasse und

$$U \times a = \{x \circ a \mid x \in U\}$$

Rechtsnebenklasse von U.

Beispiele:

1. Sei $G = (\mathbb{Z}, +)$ die additive Gruppe der ganzen Zahlen und sei

$$4\mathbb{Z} = \{\ldots, -8, -4, 0, 4, 8, 12, \ldots, 4n, \ldots; n \in \mathbb{Z}\}.$$

Die Gruppe $U = (4\mathbb{Z}, +)$ ist eine Untergruppe von G.

Check: Wir wenden das Kriterium an, dass U eine Untergruppe ist, genau dann, wenn $a^{-1} \circ b \in U$ für alle $a, b \in U$.

Seien also $a, b \in 4\mathbb{Z}$. Dann ist

$$a = 4n, n \in \mathbb{Z} \text{ und } b = 4m, m \in \mathbb{Z}.$$

Das Inverse, a^{-1} ist dann $a^{-1} = -4n \in 4\mathbb{Z}$ und

$$a^{-1} \circ b = (-4n) + 4m = 4(m - n) \in 4\mathbb{Z}, m, n \in \mathbb{Z}.$$

Da a, b beliebige Elemente in U sind, ist gezeigt, dass U eine Untergruppe von G ist.

Die Untergruppe $(4\mathbb{Z}, +)$ von $(\mathbb{Z}, +)$ hat die drei folgenden Nebenklassen:

$$4 \cdot \mathbb{Z} + 1 = \{x \mid x = 4y + 1, y \in \mathbb{Z}\}$$
$$4 \cdot \mathbb{Z} + 2 = \{x \mid x = 4y + 2, y \in \mathbb{Z}\}$$
$$4 \cdot \mathbb{Z} + 3 = \{x \mid x = 4y + 3, y \in \mathbb{Z}\}$$

2. Die Untergruppe

$$U = (\{-1, +1\}, \times)$$

der multiplikativen Gruppe $(\mathbb{Q}\backslash\{0\}, \times)$ besitzt unendlich viele Nebenklassen, nämliche $\{-x, x\}$ für alle $x \in \mathbb{Q} \setminus \{0\}, x > 0$.

3. Wir betrachten die Untergruppe

$$(\{\sigma_0, \sigma_1, \sigma_2\}, \circ)$$

der symmetrischen Gruppe $\mathcal{S}_3$. Offensichtlich gilt

$$\{\sigma_0, \sigma_1, \sigma_2\} \circ \sigma = \{\sigma_0, \sigma_1, \sigma_2\} \quad \forall\, \sigma \in \{\sigma_0, \sigma_1, \sigma_2\}$$

da $\{\sigma_0, \sigma_1, \sigma_2\}$ eine Gruppe ist. Verknüpft man die anderen drei Elemente σ_3, σ_4 und σ_5 mit den Elementen dieser Untergruppe, dann erhalten wir:

$$\{\sigma_0, \sigma_1, \sigma_2\} \circ \sigma_3 = \{\sigma_3, \sigma_5, \sigma_4\}$$
$$\{\sigma_0, \sigma_1, \sigma_2\} \circ \sigma_4 = \{\sigma_4, \sigma_5, \sigma_3\}$$
$$\{\sigma_0, \sigma_1, \sigma_2\} \circ \sigma_5 = \{\sigma_5, \sigma_4, \sigma_3\}$$

Wir konstatieren: Alle Nebenklassen der Untergruppe $(\{\sigma_0, \sigma_1, \sigma_2\}, \circ)$ mit den anderen Elementen der symmetrischen Gruppe $\mathcal{S}_3$ sind identisch. Weiterhin gilt in diesem Beispiel, dass die beiden Nebenklassen von $(\{\sigma_0, \sigma_1, \sigma_2\}\circ)$ nämlich:

- die Untergruppe $(\{\sigma_0, \sigma_1, \sigma_2\}, \circ)$
- die Menge $\{\sigma_3, \sigma_4, \sigma_5\}$

die gleiche Anzahl von Elementen hat.

Theorem: (Satz von LAGRANGE)

Sei $G = (M, \times)$ eine endliche Gruppe und $G_U = (U, \times)$ Untergruppe von G. Die Relation

$$\sim \subseteq M \times M$$

sei definiert durch:

$$a \sim b \iff b \times a^{-1} \in U.$$

Dann gilt:

① $\sim$ ist eine Äquivalenzrelation auf M

② Für jedes $a \in M$ ist $U \times a = [a]_\sim$, i.e. die Äquivalenklasse, die a enthält ist gleich der Rechtsnebenklasse $U \times a$.

③ $\mid U \mid \mid\mid M \mid$, i.e. die Ordnung von U ist eine Teiler der Ordnung von M.

Beweis:

① Um zu zeigen, dass $\sim$ eine Äquivalenzrelation ist, sind drei Eigenschaften zu verifizieren:

(a) Reflexivität
Zu zeigen ist $a \sim a$.

Für alle $a \in G$ gilt: $a \times a^{-1} = 1_G \in U$, i.e. für alle $a \in G$ ist $a \sim a$, $\sim$ ist daher reflexiv.

(b) Symmetrie
Zu zeigen ist, wenn $a \sim b$ dann ist auch $b \sim a$.

Da $a \sim b$ folgt, daß $b \times a^{-1} \in U$ gilt. Da U Untergruppe ist, ist auch das dazu inverse Element in U, i.e.

$$(b \times a^{-1})^{-1} = a \times b^{-1} \in U$$

und darus folgt, dass $b \sim a$ gilt.

(c) Transitivität
Zu zeigen ist: Wenn $a \sim b$ und $b \sim c$, dann ist auch $a \sim c$.

Wenn $a \sim b$ und $b \sim c$, dann folgt: $b \times a^{-1} \in U$ und $c \times b^{-1} \in U$. Da U Untergruppe ist, ist U abgeschlossen, dies impliziert insbesondere, daß

$$(c \times b^{-1}) \times (b \times a^{-1}) = c \times a^{-1} \in U$$

und damit $a \sim c$.

②

③

Die Untergruppen und die Ordnungen sind für zyklische Gruppen leicht zu beschreiben. Die relevanten Tatsachen sind in dem folgenden Satz zusammengefaßt.

Satz [144, Theorem 1.15]

1. Jede Untegruppe einer zyklischen Gruppe ist zyklisch.
2. In eine endlichen zyklischen Gruppe $< a >$ der Ordnung m erzeugt das Element a^k eine Untergruppe der Ordnung $m/\mathtt{ggT}(k, m)$.
3. Wenn d ein positiver Teiler der Ordnung m einer endlichen zyklischen Gruppe $< a >$ ist, dann enthält $< a >$ genau eine Untergruppe mit Index d. Für jeden positiven Teiler f von m, enthält $< a >$ genau eine Untergruppe der Ordnung f.
4. Sei f ein positiver Teiler der Ordnung einer endlichen zyklischen Gruppe $< a >$. Dann enthält $< a > \Phi(f)$ Elemente der Ordnung f. Hier bezeichnet $\Phi(f)$ die EULERsche Totientenfunktion und ist die Anzahl der ganzen Zahlen n mit $1 \leq n \leq f$, die coprim zu f sind.
5. Eine endliche zyklische Gruppe $< a >$ der Ordnung m enthält $\Phi(m)$ Erzeuger, i.e. Elemente a^r, so dass $< a^r >=< a >$. Die Erzeuger sind die Potenzen a^r mit $\mathtt{ggT}(r, m) = 1$.

Beweis:

Zu 1: Sei H eine nichttrivale Untergruppe der zyklischen Gruppe $< a >$, i.e. H besteht nicht aus dem Einselement, $H \neq \{e\}$. Falls $a^n \in H$, dann ist gemäß der Definition der Untergruppe auch das Inverse in H, i.e. $a^{-n} \in H$. Also enthält H wenigstens eine Potenz von a mit einem positiven Exponenten. Sei d der kleinste positive Exponent, so dass $a^d \in H$ und sei $a^s \in H$. Teilt man d durch s, dann folgt

$$s = qd + r, 0 \leq r < d, \quad q, r \in \mathbb{Z}.$$

Dann folgt:

$$a^s (a^{-d})^q = a^{s-dq} = a^r \in H,$$

was der Minimalität von d widerspricht, es sei denn $r = 0$. Daher sind die Exponenten aller Potenzen von a, die zu H gehören durch d teilbar, also ist $H =< a^d >$, i.e. H ist zyklisch.

Kapitel 17

Sätze von Fermat und Euler

Zwei Sätze der Zahlentheorie spielen eine sehr wichtige Rolle in den mathematischen Grundlagen von Public–Key Verfahren. Dies ist einmal der FERMATsche Satz und die Verallgemeinerung davon, das EULERsche Theorem.

17.1 Der kleine Fermatsche Satz

Wir betrachten als einführende Motivation die folgenden Kongruenzen:

$$\begin{aligned} 3^4 \bmod 5 &\equiv 1 \bmod 5 \\ 12^4 \bmod 5 &\equiv 1 \bmod 5 \\ 6^6 \bmod 7 &\equiv 1 \bmod 7 \\ 2^{10} \bmod 11 &\equiv 1 \bmod 11 \\ 3^{10} \bmod 11 &\equiv 1 \bmod 11 \\ 11^{10} \bmod 11 &\equiv 1 \bmod 11 \\ 6^{12} \bmod 13 &\equiv 1 \bmod 13 \end{aligned}$$

Diese Liste kann (fast) beliebig fortgesetzt werden. Wir erkennen folgendes Schema: Der Modulus ist stets eine Primzahl p, hier 5,7,11,13. Eine beliebige Zahl, die nicht durch p geteilt wird, wird potenziert mit $p-1$. Das Resultat ist dann stets, dass diese Potenz kongruent 1 modulo p ist.

Moderne Kryptosysteme — darunter insbesondere die Public–Key Verfahren — verwenden große Primzahlen zur Konstruktion des Schlüssels. Typischerweise sind dies 150stellige Zahlen. Die Sicherheit dieser Verfahren hängt entscheidend davon ab, dass die gewählten Zahlen auch tatsächlich Primzahlen sind. Daher besteht ein wichtiger Zwischenschritt dieser Kryptosysteme darin, zu testen, ob die gewählte Zahl auch wirklich prim ist. Diese Tests heißen **Primzahltests**. Alle bekannten Primzahltests basieren letztendlich auf einem Satz von PIERRE DE FERMAT ([172]). Dieses Theorem sagt aus, dass für jede Primzahl p und jede positive ganze Zahl a, die coprim zu p ist, der Ausdruck $a^p - a$ ein Vielfaches von p ist.[1]

[1] Eine teifgehende Einführung in Primzahltests findet man in [219].

Theorem [11]:

Kleiner Fermatscher Satz

Falls p eine Primzahl ist und a ist eine positive ganze Zahl, die nicht durch p geteilt wird, dann gilt:

$$a^{p-1} \bmod p \equiv 1 \bmod p$$

Für die Durchführung der Primzahltests wird das FERMATsche Theorem in der negierten Form benutzt: Falls für irgendwelche zufällig gewählte Zahlen a der Ausdruck $a^p - a$ kein Vielfachew von p ist, kann die Zahl p keine Primzahl sein.

Beweis:

Wir wissen, wenn p eine Primzahl ist, dann ist die Menge

$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$$

eine kommutative Gruppe unter der Multiplikation. Die Abgeschlossenheit unter Multiplikation impliziert insbesondere, dass man alle Elemente von $\mathbb{Z}_p^*$ mit a mod p multiplizieren kann. Dadurch werden alle Elemente von $\mathbb{Z}_p^*$ eventuell in einer anderen Reihenfolge reproduziert. Daher sind

$$\{a \bmod p, 2a \bmod p, \ldots, (p-1) \bmod p\}$$

die Elemente von $\mathbb{Z}_p$ in einer anderen Reihenfolge. Multipliziert man diese Elemente, erhält man:

$$\begin{aligned} a \cdot 2a \cdot \ldots \cdot (p-1)a &\equiv \{(a \bmod p) \cdot (2a \bmod p) \cdot \ldots \cdot ((p-1)a \bmod)\} \bmod p \\ &\equiv (p-1)! \bmod p. \end{aligned}$$

Andererseits haben wir:

$$a \cdot 2a \cdot 3a \cdot \ldots \cdot (p-1)a = (p-1)!a^{p-1}$$

Daher

$$(p-1)!a^{p-1} \equiv (p-1)! \bmod p.$$

Da der Term $(p-1)!$ coprim zu p ist, können wir den Term $(p-1)!$ kürzen und erhalten:

$$a^{p-1} \equiv 1 \bmod p.$$

Beispiel [17.42]

Sei $a = 7$ und $p = 19$. Wir zeigen, dass $7^{18} \equiv 1 \bmod 19$. Zunächst gilt:

$$7^2 = 49 \equiv 11 \bmod 19,$$

daher ist

$$7^4 = (7^2)^2 \equiv 121 \bmod 19 \equiv 7 \bmod 19.$$

Für diesen Schritt verwenden wir die folgende Eigenschaft der Multiplikation auf $GF(n)$:

$$a \cdot b \bmod n = [(a(\bmod n)) \cdot (b(\bmod n))](\bmod n)$$

mit $a = b = 7^2$ und $n = 19$. Wiederholt man dies, erhält man:

$$7^8 = (7^4) \cdot (7^4) \equiv 49 \bmod 19 \equiv 11 \bmod 19$$

und

$$7^{16} = (7^8) \cdot (7^8) \equiv 121 \bmod 19 \equiv 7 \bmod 19.$$

Damit (für $a = 7, p - 1 = 18$):

$$7^{18} = (7^{16}) \cdot (7^2) \equiv 7 \cdot 11 \bmod 19 \equiv 1 \bmod 19$$

Bemerkung:

Die Kongruenz

$$a^{p-1} \equiv 1 \bmod p$$

ist äquivalent zu

$$a^p - a \equiv 0 \bmod p.$$

Aufgrund der modularen Arithmetik haben wir:

$$\begin{aligned} a^{p-1} \cdot a &= a^p \\ &\equiv a \bmod p \end{aligned}$$

was das gleiche ist wie

$$a^p - a \equiv 0 \bmod p.$$

17.2 Der Satz von Euler

Eine Verallgemeinerung des kleinen FERMATschen Satzes ist der Satz von EULER.

Theorem [12]:

Satz von Euler

Sei $n \geq 2$ eine natürliche Zahl. Dann gilt für jede ganze Zahl a mit $\mathrm{ggT}(a, n) = 1$:

$$a^{\phi(n)} \equiv 1 \bmod n.$$

Beweis:

Die obige Gleichung ist korrekt, falls n eine Primzahlzahl ist. In diesem Fall ist die Totientenfunktion $\phi(n) = n - 1$ und die Aussage des Satzes von EULER reduziert sich auf den kleinen Satz von FERMAT.

Im allgemeinen Fall, wenn n also keine Primzahl ist, zeigen wir die Aussage wie folgt. Die Totientenfunktion $\phi(n)$ ist die Anzahl der positiven ganzen Zahlen kleiner n, die coprim zu n sind. Diese Menge von ganzen Zahlen bezeichnen wir mit:

$$R = \{x_1, x_2, \ldots, x_{\phi(n)}\}.$$

Wir multiplizieren nun jedes Element dieser Menge mit $a \bmod n$ und erhalten die Menge:

$$S = \{(ax_1 \bmod n), (ax_2 \bmod n), \ldots, (ax_{\phi(n)} \bmod n)\}.$$

Die Menge S ist nichts anderes als eine Permutation der Menge R aufgrund der folgenden Argumente:

1. Da a coprim zu n ist und sämtliche x_i coprim zu n sind, müssen die Produkte $a \cdot x_i$ ebenfalls coprim zu n sein. Dies impliziert, dass die Menge S alle ganzen Zahlen enthält, die kleiner als n sind und zu n coprim sind.

2. Es gibt in S keine Duplikate. Dies erkennt man leicht aufgrund der Regel: Falls

   $$(a \cdot b) \equiv (a \cdot c) \bmod n$$

   dann ist

   $$b \equiv c \bmod n$$

   falls a coprim zu n ist. Damit erhalten wir, falls $ax_i \bmod n = ax_j \bmod n$ dann ist $x_i = x_j$.

Damit erhalten wir:

$$\prod_{i=1}^{\phi(n)} (ax_i \bmod n) = \prod_{i=1}^{\phi(n)} x_i$$

i.e. multipliziert man alle Zahlen in S, dann erhält man das gleiche Ergebnis wie die Multiplikation aller Zahlen in R, da beide Mengen die gleichen Zahlen enthalten, allerdings in unterschiedlicher Reihenfolge. Diese Gleichung besagt, dass beide Produkte auf die gleiche Zahl führen. Dies wiederum impliziert, wenn man beide Seiten durch n dividiert, haben linke und rechte Seiten den gleichen Rest. Mit anderen Worten, $\prod_{i=1}^{\phi(n)} (ax_i \bmod n)$ liegt in der gleichen Restklasse wie $\prod_{i=1}^{\phi(n)} x_i$, *i.e.*

$$\left(\prod_{i=1}^{\phi(n)} (ax_i \bmod n)\right) \bmod n \equiv \left(\prod_{i=1}^{\phi(n)} x_i\right) \bmod n.$$

Wendet man die Regel

$$[(a(\bmod n)) \cdot (b(\bmod n))](\bmod n) \equiv (a \cdot b)(\bmod n)$$

an, dann erhält man:

$$\prod_{i=1}^{\phi(n)} ax_i \equiv \left(\prod_{i=1}^{\phi(n)} x_i\right) \bmod n.$$

Die linke Seite enthält $\phi(n)$ mal die Zahl a, wir können daher schreiben:

$$a^{\phi(n)} \cdot \left[\prod_{i=1}^{\phi(n)} x_i\right] \equiv \left(\prod_{i=1}^{\phi(n)} x_i\right) \bmod n.$$

Nun sind alle Zahlen x_i coprim zu n, dann ist das Produkt dieser Zahlen ebenfalls coprim zu n. Da $\mathtt{ggT}(n, \prod x_i) = 1$ erhalten wir:

$$a^{\phi(n)} \equiv 1 \bmod n$$

Damit ist der Satz von Euler bewiesen.

Eine alternative Form des Satzes von Euler ist sehr zweckmäßig. Multiplizieren wir mit a, erhalten wir:

$$a^{\phi(n)+1} \equiv a \bmod n$$

Das folgende Korollar des Satzes von Euler ist essentiell, um die Korrektheit des RSA Algorithmus zu zeigen.

Korollar:

Seien p und q zwei Primzahlen, $n = p \cdot q$ und m zwei ganze Zahlen mit $0 < m < n$ dann gilt

$$m^{\phi(n)+1} = m^{(p-1)(q-1)+1} \equiv m \bmod n. \tag{17.1}$$

Beweis:

Zunächst sei erwähnt, dass das Korollar auch dann gilt, wenn m und n *nicht* coprim sind. Falls $\texttt{ggT}(m,n) = 1$ — *i.e.* m und n sind relativ prim — dann gilt Gl. (17.1) aufgrund des Satzes von Euler.

Betrachte nun den Fall, dass gilt: $\texttt{ggT}(m,n) \neq 1$ mit $n = p \cdot q$. Dies impliziert, dass m entweder ein Vielfaches von p oder ein Vielfaches von q ist[2]. Angenommen, m ist ein Vielfaches von p, dann ist $m = cp$ für eine positive ganze Zahl c. Dann muß $\texttt{ggT}(m,q) = 1$ gelten, da sonst m ein Vielfaches von p und ein Vielfaches von q ist, was aber der Annahme $0 < m < n = pq$ widerspricht. Falls $\texttt{ggT}(m,q) = 1$, können wir den Satz von Euler anwenden und erhalten:

$$m^{\phi(q)} \equiv 1 \bmod q$$

Aus den grundlegenden Eigenschaften der modularen Arithmetik wissen wir, wenn die Kongruenz

$$a \equiv b \bmod m$$

gilt, dann gilt diese Kongruenz auch für jede Potenz:

$$a^n \equiv b^n \bmod m.$$

Damit erhalten wir:

$$\begin{aligned}(m^{\phi(q)})^{\phi(p)} &\equiv 1^{\phi(p)} \bmod q \\ &= 1 \bmod q\end{aligned}$$

Da gemäß Annahme p und q Primzahlen sind und $n = p \cdot q$, dann erfüllt die Eulersche Totientenfunktion die Beziehung:

$$\phi(n) = \phi(p) \cdot \phi(q)$$

und damit:

$$m^{\phi(n)} \equiv 1 \bmod q$$

Daher gibt es eine Zahl $k \in \mathbb{Z}$ mit:

$$m^{\phi(n)} = 1 + k \cdot q.$$

Multipliziert man beide Seiten mit $m = c \cdot p$ erhält man:

$$m^{\phi(n)+1} = m + m \cdot k \cdot q = m + c \cdot p \cdot k \cdot q = m + kcn.$$

[2] Dies ist deshalb der Fall, da n das Produkt der beiden Primzahlen p und q ist, i.e. die einzigen nichttrivialen Teiler von n sind p und q. Wenn $\texttt{ggT}(m,n) \neq 1$, dann muss p oder q auch Teiler von m sein.

Daher:

$$m^{\phi(n)+1} \equiv m \bmod n.$$

Alternative Formen dieses Korollars sind

$$\begin{aligned}\left(m^{\phi(n)}\right)^k &\equiv 1 \bmod n \\ m^{k\phi(n)} &\equiv 1 \bmod n \\ m^{k\phi(n)+1} &= m^{k(p-1)(q-1)+1} \equiv m \bmod n\end{aligned}$$

17.3 Quadratische Reste

Ein weiteres Konzept der Zahlentheorie, das in vielen Kryptosystemen — unter anderem bei der Untersuchung von Kryptosystemen auf elliptischen Kurven über einem endlichen Körper oder im FIAT–SHAMIR Protokoll — benötigt wird, sind die quadratischen Reste.[3]

Definition:

Ein $a \in \mathbb{N}$ mit $\mathrm{ggT}(a,n) = 1$ heißt *quadratischer Rest modulo n*, wenn ein $x \in \mathbb{N}$ existiert mit

$$x^2 \bmod n \equiv a \bmod n.$$

Ist die Bedingung nicht erfüllt, nennt man a ein **quadratischer Nichtrest modulo n**. Man setzt

$$R_n = \{a \in \mathbb{N} \mid a \text{ ist quadratischer Rest mod } n, a \in \mathbb{Z}_n^*\}.$$

Ist a ein quadratischer Rest modulo n, dann nennt man x mit $x^2 \equiv a \bmod n$ *Quadratwurzel modulo n*.

Beispiele:

Sei $n = 11$. Wir suchen alle quadratischen Reste modulo 11. Wir berechnen dazu sukzessive:

$$\begin{aligned}
1^2 &\equiv 1 \bmod 11\\
2^2 &\equiv 4 \bmod 11\\
3^2 &\equiv 9 \bmod 11\\
4^2 &\equiv 5 \bmod 11\\
5^2 &\equiv 3 \bmod 11\\
6^2 &\equiv 3 \bmod 11\\
7^2 &\equiv 5 \bmod 11\\
8^2 &\equiv 9 \bmod 11\\
9^2 &\equiv 4 \bmod 11\\
10^2 &\equiv 1 \bmod 11
\end{aligned}$$

Damit ist

$$R_{11} = \{1, 3, 4, 5, 9\}$$

die Menge der quadratischen Reste mod 11 und die Zahlen 2,6,7,8 und 10 sind die Nichtreste modulo 11.

Aus diesem Beispiel kann man folgende Sachverhalte ablesen:

[3] Wir folgen hier der Darstellung von [218, Kap. 9].

- Sieht man sich den quadratischen Rest 3 etwas näher an, dann erkennt man aus obiger Tabelle, dass es zu dem quadratischen Rest 3 genau zwei verschiedene Quadratwurzeln modulo 11 gibt, nämlich $x = 5$ und $x = -6$. Gleiches gilt für alle anderen quadratischen Reste:
  - Der quadratische Rest $a = 1$ hat die Wurzeln $1, 10$.
  - Der quadratische Rest $a = 4$ hat die Wurzeln $2, 9$.
  - Der quadratische Rest $a = 5$ hat die Wurzeln $4, 7$.
  - Der quadratische Rest $a = 9$ hat die Wurzeln $3, 8$.
- Wenn a ein quadratischer Rest ist, dann auch a mod 11.
- Es gibt $5 = (11 - 1)/2$ quadratische Reste modulo 11
- Es gibt $5 = (11 - 1)/2$ quadratische Nichtreste modulo 11

Als ein weiteres Beispiel betrachten wir den Fall $n = 13$ und suchen alle Quadratwurzeln modulo 13.

$$
\begin{aligned}
1^2 &\equiv 1 \bmod 13 \\
2^2 &\equiv 4 \bmod 13 \\
3^2 &\equiv 9 \bmod 13 \\
4^2 &\equiv 3 \bmod 13 \\
5^2 &\equiv 12 \bmod 13 \\
6^2 &\equiv 10 \bmod 13 \\
7^2 &\equiv 10 \bmod 13 \\
8^2 &\equiv 12 \bmod 13 \\
9^2 &\equiv 3 \bmod 13 \\
10^2 &\equiv 9 \bmod 13 \\
11^2 &\equiv 4 \bmod 13 \\
12^2 &\equiv 1 \bmod 13
\end{aligned}
$$

Damit ist

$$R_{13} = \{1, 3, 4, 9, 10, 12\}$$

die Menge der quadratischen Reste modulo 13 und die Zahlen 2,5,6,7,8,11 sind die quadratischen Nichtreste. Auch hier gilt:

- Es gibt $6 = (13 - 1)/2$ quadratische Reste modulo 13
- Es gibt $6 = (13 - 1)/2$ quadratische Nichtreste modulo 13

Wir werden unten das EULERsche Kriterium zur Bestimmung quadratischer

Reste untersuchen. Dazu betrachten wir die folgenden Potenzen:

$$
\begin{aligned}
1^{\frac{p-1}{2}} \bmod p &= 1^6 \bmod 13 \equiv 1 \bmod 13\\
2^{\frac{p-1}{2}} \bmod p &= 2^6 \bmod 13 \equiv 12 \bmod 13 \equiv -1 \bmod 13\\
3^{\frac{p-1}{2}} \bmod p &= 3^6 \bmod 13 \equiv 1 \bmod 13\\
4^{\frac{p-1}{2}} \bmod p &= 4^6 \bmod 13 \equiv 1 \bmod 13\\
5^{\frac{p-1}{2}} \bmod p &= 5^6 \bmod 13 \equiv 12 \bmod 13 \equiv -1 \bmod 13\\
6^{\frac{p-1}{2}} \bmod p &= 6^6 \bmod 13 \equiv 12 \bmod 13 \equiv -1 \bmod 13\\
7^{\frac{p-1}{2}} \bmod p &= 7^6 \bmod 13 \equiv 12 \bmod 13 \equiv -1 \bmod 13\\
8^{\frac{p-1}{2}} \bmod p &= 8^6 \bmod 13 \equiv 12 \bmod 13 \equiv -1 \bmod 13\\
9^{\frac{p-1}{2}} \bmod p &= 9^6 \bmod 13 \equiv 1 \bmod 13\\
10^{\frac{p-1}{2}} \bmod p &= 10^6 \bmod 13 \equiv 1 \bmod 13\\
11^{\frac{p-1}{2}} \bmod p &= 11^6 \bmod 13 \equiv 12 \bmod 13 \equiv -1 \bmod 13\\
12^{\frac{p-1}{2}} \bmod p &= 12^6 \bmod 1 \equiv 12 \bmod 13
\end{aligned}
$$

Hieraus erkennt man folgenden Sachverhalt:

- ☞ Wenn a ein quadratischer Rest modulo 13 ist, dann ist

$$a^{\frac{13-1}{2}} \bmod 13 \equiv 1 \bmod 1$$

- ☞ Wenn a ein quadratischer Nichtrest modulo 13 ist, dann ist

$$a^{\frac{13-1}{2}} \bmod 13 \equiv 1 \bmod -1$$

Wir betrachten zunächst quadratische Reste modulo einer Primzahl p. Der folgende Satz sagt aus, dass es für jeden quadratischen Rest genau zwei Lösungen gibt, i.e. zwei Quadratwurzeln modulo n oder – im Fall von quadratischen Nichtresten – keine Lösung gibt.

Satz:

Sei p eine Primzahl und $a \in \mathbb{Z}_p^*$. Dann hat die Kongruenz

$$x^2 \bmod p \equiv a \bmod p$$

für $a \in R_p$ zwei, für $a \notin R_p$ keine Lösungen.

Beweis:

Sei $a \in R_p$. Dann gibt es nach Definition eine Lösung $x_1 \in \mathbb{N}$ mit

$$x_1^2 \bmod p \equiv a \bmod p$$

Dann ist aber wegen

$$\begin{aligned}(p-x_1)^2 \bmod p &\equiv (p^2 - 2x_1p + x_1^2) \bmod p \\ &\equiv x_1^2 \bmod p \\ &\equiv a \bmod p\end{aligned}$$

die Zahl $p - x_1$ ebenfalls Lösung. Die beiden Lösungen x_1 und $p - x_1$ sind aber verschieden, sonst folgt aus

$$p - x_1 = x_1$$

der Widerspruch $p = 2x_1$. Dies ist ein Widerspruch zur Annahme, daß p eine Primzahl ist. Weitere Lösungen existieren nicht, da das Polynom

$$f(x) = x^2 - a$$

in $\mathbb{Z}_p$ höchstens zwei Nullstellen hat. Für $a \notin R_p$ gibt es per Definition keine Lösung.

Der folgende Satz gibt an, wieviele quadratische Reste modulo einer Primzahl p existieren.

Satz:

Sei $p > 2$ eine Primzahl. Dann existieren $\frac{p-1}{2}$ quadratische Reste modulo p und $\frac{p-1}{2}$ quadratische Nichtreste modulo p.

Beweis:

Die Zahlen

$$1^2 \bmod p, 2^2 \bmod p, \ldots, \left(\frac{p-1}{2}\right)^2 \bmod p$$

sind offenbar quadratische Reste. Diese Reste sind alle voneinander verschieden, da jede Zahl

$$y^2 \bmod p, y \in \left\{1, 2, \ldots, \frac{p-1}{2}\right\}$$

genau die beiden verschiedenen Lösungen y und $p - y$ hat. Da für jede Zahl $a \in R_p$ genau eine ihrer Wurzeln x_1 und $p - x_1$ in der Menge

$$\left\{1, 2, \ldots, \frac{p-1}{2}\right\}$$

liegt, existieren keine weiteren quadratischen Reste. Alle $a \in R_p$ liegen daher in der Menge

$$1^2 \bmod p, 2^2 \bmod p, \ldots, \left(\frac{p-1}{2}\right)^2 \bmod p$$

Um den folgenden Satz zu motivieren betrachten wir nochmals die quadratischen Reste modulo 11. Wir berechnen sukzessive die Potenzen

$$x^{\frac{p-1}{2}} \bmod p \qquad \text{für } p = 11, x \in \mathbb{Z}_{11}$$

$$\begin{aligned}
1^5 \bmod 11 &\equiv 1 \bmod 11 \\
2^5 \bmod 11 &\equiv 10 \bmod 11 \\
3^5 \bmod 11 &\equiv 1 \bmod 11 \\
4^5 \bmod 11 &\equiv 1 \bmod 11 \\
5^5 \bmod 11 &\equiv 1 \bmod 11 \\
6^5 \bmod 11 &\equiv 10 \bmod 11 \\
7^5 \bmod 11 &\equiv 10 \bmod 11 \\
8^5 \bmod 11 &\equiv 10 \bmod 11 \\
9^5 \bmod 11 &\equiv 1 \bmod 11 \\
10^5 \bmod 11 &\equiv 10 \bmod 11
\end{aligned}$$

Mit anderen Worten, dann, wenn x ein quadratischer Rest ist, ist $x^5 \bmod 11 \equiv 1 \bmod 11$, sonst gilt

$$x^5 \bmod 11 \equiv 10 \bmod 11$$

Diese Beobachtung gilt allgemein, was folgender Satz aussagt:

Satz:

Sei $p > 2$ eine Primzahl und $a \in \mathbb{Z}_p^*$. Dann folgt

$$a^{\frac{p-1}{2}} \bmod p = \begin{cases} 1 & \text{falls } a \in R_p \\ p-1 & \text{sonst} \end{cases}.$$

Beweis:

Nach dem kleinen Satz von FERMAT gilt für p Primzahl:

$$\left(a^{p-1} - 1\right) \bmod p \equiv 0 \bmod p$$

Da p eine ungerade Zahl ist, folgt:

$$\left(a^{\frac{p-1}{2}} - 1\right) \cdot \left(a^{\frac{p-1}{2}} + 1\right) \bmod p = \left(a^{p-1} - 1\right) \bmod p \equiv 0 \bmod p$$

Mit anderen Worten, die Primzahl p ist Teiler von $a^{\frac{p-1}{2}} - 1$ oder von $a^{\frac{p-1}{2}} + 1$. Da die Differenz der beiden Terme 2 ist, kann p nicht beide Terme gleichzeitig teilen. Damit gilt aber:

$$a^{\frac{p-1}{2}} \bmod p \equiv 1 \bmod p$$

oder

$$a^{\frac{p-1}{2}} \bmod p \equiv -1 \bmod p \equiv (p-1) \bmod p$$

Falls nun $a \in R_p$, dann existiert ein $x \in \mathbb{Z}_p^*$ mit $a \equiv x^2 \bmod p$. In diesem Fall gilt daher

$$a^{\frac{p-1}{2}} \bmod p \equiv (x^2)^{\frac{p-1}{2}} \bmod p \equiv x^{p-1} \bmod p \equiv 1 \bmod p$$

Daraus folgt, dass die $\frac{p-1}{2}$ quadratischen Reste Lösungen der Gleichung

$$x^{\frac{p-1}{2}} \bmod p \equiv 1 \bmod p$$

sind. Da diese Gleichung im Körper $\mathbb{Z}_p^*$ höchstens $\frac{p-1}{2}$ Lösungen hat, sind die $\frac{p-1}{2}$ Nichtreste Lösungen der Gleichung

$$x^{\frac{p-1}{2}} \bmod p \equiv p-1 \bmod p$$

Wir diskutieren nun das Problem zu bestimmen, ob eine gegebene Zahl a ein quadratischer Rest modulo p ist.

Theorem: (EULERsches Kriterium)

Sei $p > 2$ Primzahl. a ist ein quadratischer Rest modulo p genau dann, wenn gilt:

$$a^{\frac{p-1}{2}} \equiv 1 \bmod p$$

Für eine Primzahl p gibt es nach dem ersten Satz dieses Abschnitts genau zwei Quadratwurzeln. Für spezielle Primzahlen p mit der Eigenschaft

$$p \equiv 3 \bmod 4$$

können diese leicht berechnet werden, denn falls

$$p \equiv 3 \bmod 4$$

gilt, ist 4 Teiler von $p+1$. Wegen $a \in R_p$ folgt nach obigem Satz:

$$a^{\frac{p-1}{2}} \bmod p \equiv 1 \bmod p$$

Damit folgt:

$$\begin{aligned}
\left(a^{\frac{p+1}{4}}\right)^2 \bmod p &\equiv a^{\frac{p+1}{2}} \bmod p \\
&\equiv a^{\frac{p-1}{2}+1} \bmod p \\
&\equiv a \cdot a^{\frac{p-1}{2}} \bmod p \\
&\equiv a \cdot 1 \bmod p \\
&\equiv a \bmod p
\end{aligned}$$

Damit ist

$$x_1 = a^{\frac{p+1}{4}} \bmod p$$

eine Lösung von

$$x^2 \equiv a \bmod p$$

Die zweite Lösung ist $p - x_1$.

Beispiel:

Wir betrachten die Primzahl $p = 47$. Mit Hilfe des `fastexp` Algorithmus überzeugt man sich leicht, dass

$$12^{\frac{47-1}{2}} \bmod 47 \equiv 12^{23} \bmod 47 \equiv 1 \bmod 47.$$

Daher ist 12 ein quadratischer Rest modulo 47. Gesucht ist nun ein $x \in \mathbb{Z}_{47}^*$ mit

$$x^2 \equiv 12 \bmod 47.$$

Da

$$47 \equiv 3 \bmod 4$$

berechnen wir:

$$\begin{aligned} 12^{\frac{p+1}{4}} \bmod 47 &\equiv 12^{12} \bmod 47 \\ &\equiv 24 \bmod 47. \end{aligned}$$

Daher lautet die Lösung

$$x_1 = 24$$

Die zweite Lösung erhält man durch:

$$x_2 = p - x_1 = 23$$

Check:

$$24^2 \bmod 47 = 576 \bmod 47 \equiv 12 \bmod 47$$

und

$$23^2 \bmod 47 = 529 \bmod 47 \equiv 12 \bmod 47$$

17.4 Quadratwurzeln modulo n

Ziel dieses Abschnitts ist die Berechnung der Quadratwurzeln für vorgegebene quadratische Reste modulo n. Zunächst betrachten wir wieder den Fall, dass n eine Primzahl ist, anschließend den Fall, dass n das Produkt zweier Primzahlen ist.

Wie wir im Abschnitt [17.3] gezeigt haben, gibt es für jeden quadratischen Rest genau zwei Quadratwurzeln.

- Spezielle Primzahlen der Form $p \bmod 4 \equiv 3 \bmod 4$:
  Gilt speziell $4 \mid (p+1)$, dann lassen sich diese beiden Quadratwurzeln leicht berechnen. Wegen $a \in R_p$ folgt nach den EULER Kriterium

$$a^{\frac{p-1}{2}} \bmod p \equiv 1 \bmod p.$$

  Damit erhält man:

$$\begin{aligned}\left(a^{\frac{p+1}{4}}\right)^2 \bmod p &\equiv a^{\frac{p+1}{2}} \bmod p \\ &\equiv a \cdot a^{\frac{p-1}{2}} \bmod p \\ &\equiv a.\end{aligned}$$

  In diesem Fall ist daher

$$x_1 = a^{\frac{p+1}{4}} \bmod p$$

  eine Lösung (i.e. Quadratwurzel modulo p) von $x^2 \bmod p \equiv a \bmod p$. Die zweite Lösung ist dann

$$x_2 = p - x_1.$$

Beispiel:

Sei $p = 19$. Durch Berechnung von

$$16^{\frac{19-1}{2}} \bmod 19 \equiv 1 \bmod 19$$

überzeugt man sich schnell, dass $a = 16 \in R_{19}$ gilt. Wegen

$$4 \mid (19+1)$$

erhalten wir die Wurzel:

$$\begin{aligned}x_1 &= a^{\frac{p+1}{4}} \bmod p \\ &= 16^5 \bmod 19 \\ &= 4.\end{aligned}$$

Die zweite Lösung ist:

$$x_2 = p - x_1 = 15.$$

In der Tat:

$$15^2 \bmod 19 \equiv 16 \bmod 19.$$

- Allgemeine Primzahl
  Im allgemeinen ist die Bestimmung der Quadratwurzel komplizierter. In [218, pp. 160] ist ein probabilistischer Algorithmus angegeben, der mit Laufzeit $O((\log_2 p)^k)$ die Quadratwurzel modulo p berechnet.

Seien p, q zwei Primzahlen und $n = p \cdot q$. Dann gilt folgender Satz:

Theorem:

Es sei $n \in \mathbb{N}, n = p \cdot q$ mit zwei Primfaktoren p, q. Sei $a \in \mathbb{Z}_n^*$. Dann hat die Gleichung

$$x^2 \bmod n \equiv a \bmod n$$

für $a \in R_n$ genau vier Lösungen, für $a \notin R_n$ keine Lösung.

Beweis:

Für $a \notin R_n$ kann es nach Definition eines quadratischen Restes keine Lösung geben. Für $a \in R_n$ sei x eine Lösung der Gleichung

$$x^2 \bmod n \equiv a \bmod n \quad \text{oder} \quad x^2 \bmod \quad p \cdot q \equiv a \bmod p \cdot q.$$

Dann folgt:

$$x^2 \bmod p \equiv a \bmod p \tag{17.2}$$
$$x^2 \bmod q \equiv a \bmod q \tag{17.3}$$

Also: Wenn $n = p \cdot q$, p, q zwei Primzahlen und

$$x^2 \bmod n \equiv a \bmod n$$

dann folgt

$$x^2 \bmod p \equiv a \bmod p$$
$$x^2 \bmod q \equiv a \bmod q$$

Dies zeigt man folgendermaßen: Die Kongruenz

$$x^2 \bmod n \equiv a \bmod n$$

bedeutet, es gibt ein $k \in \mathbb{Z}$ mit

$$x^2 - a = k \cdot n = k \cdot (p \cdot q).$$

Daraus folgt aber:

$$x^2 - a = (k \cdot p) \cdot q \quad \Longleftrightarrow \quad x^2 \bmod q \equiv a \bmod q$$

und:

$$x^2 - a = (k \cdot q) \cdot p \quad \Longleftrightarrow \quad x^2 \bmod p \equiv a \bmod p.$$

Also: $x \bmod p$ ist eine Lösung der Gl. (17.2) und $x \bmod q$ ist Lösung der Gleichung (17.3).

Die Gleichung (17.2) besitzt die beiden Lösungen

$$x_1, p - x_1 \in \mathbb{Z}_p^*$$

und Gleichung (17.3) die beiden Lösungen

$$x_2, q - x_2 \in \mathbb{Z}_q^*.$$

Gemeinsame Lösungen ergeben sich nach den Chinesischen Restesatz (siehe Kapitel [15.4]) durch

$$x \bmod p = x'$$
$$x \bmod q = x''.$$

Hierbei steht $x' \in \{x_1, p - x_1\}$ und $x'' \in \{x_2, q - x_2\}$. Die Kombination der jeweils zwei Lösungen ergibt vier Gleichungssysteme, die genau eine Lösung in $\mathbb{Z}_n$ besitzen, hier in $\mathbb{Z}_n^*$.

Aus dem konstruktiven Beweis des Satzes kann direkt ein Algorithmus angegeben werden, um die Quadratwurzel modulo n zu berechnen.

Algorithmus Berechnung der Quadratwurzel modulo n

Input: $n \in \mathbb{N}, n = p \cdot q$, p, q zwei Primzahlen, $a \in R_n$.

Output: Vier verschiedene Lösungen $x \in \mathbb{Z}_n^*$ mit

$$x^2 \bmod n \equiv a \bmod n$$

Bestimme die beiden Quadratwurzeln $x_1, p - x_1$ von

$$x^2 \bmod p \equiv a \bmod p$$

Bestimme die beiden Quadratwurzeln $x_2, p - x_2$ von

$$x^2 \bmod q \equiv a \bmod q$$

Stelle die vier Gleichungssysteme auf

$$x \bmod p \equiv x_1 \bmod p$$
$$x \bmod p \equiv (p - x_1) \bmod p$$
$$x \bmod q \equiv x_2 \bmod q$$
$$x \bmod q \equiv (q - x_2) \bmod q$$

Löse diese Gleichungssysteme mit Hilfe des Chinesischen Restesatzes.

Beispiel:

Wir betrachten als Beispiel $p = 3, q = 7$, dann ist $n = 21$. Gesucht sind die vier Lösungen der quadratischen Kongruenz

$$x^2 \bmod 21 \equiv 16 \bmod 21. \tag{17.4}$$

1. Bestimme die Wurzeln der Kongruenz

   $$x^2 \bmod 3 \equiv 16 \bmod 3 \equiv 1 \bmod 3.$$

   Die Lösungen können direkt angegeben werden: $x_1 = 1, p - x_1 = 2$.

2. Bestimme die Wurzeln der Kongruenz

   $$x^2 \bmod 7 \equiv 16 \bmod 7 \equiv 2 \bmod 7.$$

   Hier sind die Lösungen $x_2 = 4, q - x_2 = 3$.

3. Wir bestimmen nun die vier gemeinsamen Lösungen mit Hilfe des Chinesischen Restesatzes. Dazu betrachten wir die vier Kongruenzen

   $$\begin{aligned} x \bmod p &\equiv x_1 \bmod p \\ x \bmod p &\equiv (p - x_1) \bmod p \\ x \bmod q &\equiv x_2 \bmod q \\ x \bmod q &\equiv (q - x_2) \bmod q, \end{aligned}$$

   die mit den expliziten Werten auf die folgenden Kongruenzenführen:

   $$x \bmod 3 \equiv 1 \bmod 3 \tag{17.5}$$
   $$x \bmod 3 \equiv 2 \bmod 3 \tag{17.6}$$
   $$x \bmod 7 \equiv 4 \bmod 7 \tag{17.7}$$
   $$x \bmod 7 \equiv 3 \bmod 7. \tag{17.8}$$

   (a) Betrachte die beiden Kongruenzen (17.5) und (17.7). Hierauf wenden wir den Chinesischen Restesatz (siehe Abschnitt [15.4]) an. Wir setzen also:

   $$a_1 = 1, a_2 = 4, m_1 = 3, m_2 = 7.$$

   Damit ist

   $$m = m_1 \cdot m_2 = 3 \cdot 7 = 21.$$

   Dann ist

   $$\begin{aligned} \xi_1 &= \frac{m}{m_1} = m_2 = 7 \\ \xi_2 &= \frac{m}{m_2} = m_1 = 3 \end{aligned}$$

   und

   $$\begin{aligned} \xi_1^{-1} \bmod m_1 &= 7^{-1} \bmod 3 \equiv 2^{-1} \bmod 3 = 2 \\ \xi_2^{-1} \bmod m_2 &= 3^{-1} \bmod 7 = 5. \end{aligned}$$

Damit erhalten wir:

$$c_1 = \xi_1(\xi_1^{-1} \bmod m_1) = 7 \cdot 2 = 14$$
$$c_2 = \xi_2(\xi_2^{-1} \bmod m_2) = 3 \cdot 5 = 15$$

und

$$z_1 = (a_1 \cdot c_1 + a_2 \cdot c_2) \bmod m_1 m_2 = 74 \bmod 21 \equiv 11 \bmod 21.$$

(b) Betrachte die beiden Kongruenzen (17.5) und (17.8). Hierauf wenden wir erneut den Chinesischen Restesatz (siehe Abschnitt [15.4]) an. Wir setzen also:

$$a_1 = 1, a_2 = 3, m_1 = 3, m_2 = 7.$$

Damit ist

$$m = m_1 \cdot m_2 = 3 \cdot 7 = 21.$$

Dann ist

$$\xi_1 = \frac{m}{m_1} = m_2 = 7$$
$$\xi_2 = \frac{m}{m_2} = m_1 = 3$$

und

$$\xi_1^{-1} \bmod m_1 = 7^{-1} \bmod 3 \equiv 2^{-1} \bmod 3 = 2$$
$$\xi_2^{-1} \bmod m_2 = 3^{-1} \bmod 7 = 5.$$

Damit erhalten wir:

$$c_1 = \xi_1(\xi_1^{-1} \bmod m_1) = 7 \cdot 2 = 14$$
$$c_2 = \xi_2(\xi_2^{-1} \bmod m_2) = 3 \cdot 5 = 15$$

und

$$z_2 = (a_1 \cdot c_1 + a_2 \cdot c_2) \bmod m_1 m_2 = 59 \bmod 21 \equiv 17 \bmod 21.$$

(c) Als nächstes untersuchen wir die beiden Kongruenzen (17.6) und (17.7). Hierauf wenden wir erneut den Chinesischen Restesatz (siehe Abschnitt [15.4]) an. Wir setzen also:

$$a_1 = 2, a_2 = 4, m_1 = 3, m_2 = 7.$$

Die Werte für ξ_1, ξ_2, c_1, c_2 bleiben wie oben. Damit

$$z_3 = (a_1 \cdot c_1 + a_2 \cdot c_2) \bmod m_1 m_2 = 88 \bmod 21 \equiv 4 \bmod 21.$$

(d) Schließlich bleiben noch die beiden Kongruenzen (17.6) und (17.8):

$$a_1 = 2, a_2 = 3, m_1 = 3, m_2 = 7.$$

Die Werte für ξ_1, ξ_2, c_1, c_2 bleiben wie oben. Damit

$$z_4 = (a_1 \cdot c_1 + a_2 \cdot c_2) \bmod m_1 m_2 = 73 \bmod 21 \equiv 10 \bmod 21.$$

Damit haben wir die vier Quadratwurzeln der quadratischen Kongruenz (17.4) gefunden:

$$z_1 = 11, z_2 = 17, z_3 = 4, z_4 = 10.$$

Check: Wir überprüfen unser Ergebnis durch Berechnen von

$$z_i^2 \bmod 21 \stackrel{!}{\equiv} 16 \bmod 21.$$

Es folgt

$$\begin{aligned} 11^2 \bmod 21 &\equiv 121 \bmod 21 \equiv 16 \bmod 21 \\ 17^2 \bmod 21 &\equiv 289 \bmod 21 \equiv 16 \bmod 21 \\ 10^2 \bmod 21 &\equiv 100 \bmod 21 \equiv 16 \bmod 21 \\ 4^2 \bmod 21 &\equiv 16 \bmod 21. \end{aligned}$$

Der obige Algorithmus berechnet die Quadratwurzeln modulo n in effektiver Laufzeit[4]. Wesentlich dabei ist, dass die beiden Primfaktoren p und q bekannt sind.

Für kryptographische Algorithmus ist der folgende Satz relevant.

Satz

Sei n das Produkt zweier Primzahlen p und q. Bei unbekannten Zahlen p und q ist das Problem, eine Quadratwurzel modulo n zu einem quadratischen Rest aus R_n zu bestimmen berechnungsmäßig äquivalent zu dem Problem, die Primfaktoren p und q von n zu berechnen.

Beweis:

Berechnungsmäßige Äquivalenz bedeutet, dass jedes der beiden Probleme unter Benutzung eines Polynomzeit-Algorithmus für das jeweils andere Problem selbst in polynomialer Zeit gelöst werden kann.

[4] Die Laufzeitbetrachtung dieses Algorithmus ist explizit in [218] untersucht.

Zunächst ist klar, dass bei bekanntem p und q mit Hilfe des obigen Algorithmus in polynomialer Zeit die vier Quadratwurzeln gefunden werden.

Umgekehrt sei A ein Polynomialzeitalgorithmus, der eine Quadratwurzel bestimmt. Zu zeigen ist, dass mit Hilfe von A die Zahl n faktorisiert werden kann. Dazu betrachten wir ein zufälliges $x \in \mathbb{Z}_n^*$ und berechnen $a = x^2 \bmod n$. Dann wendet man den Algorithmus A auf a und n an und erhält damit in polynomialer Laufzeit eine Quadratwurzel y von a mod n. Es können nun zwei Fälle eintreten:

1. $y = x$ oder $y = n - x$. In diesem Falle erhält man keine neuen Informationen, und man wählt ein neues x.

2. $y \neq x$ und $y \neq n - x$. Dann folgt:

$$(x^2 - y^2) \bmod n \equiv 0 \bmod n.$$

   Dies bedeutet aber:

$$n = pq | (x + y)(x - y)$$

   Angenommen, $n \mid (x + y)$. Da $0 \leq x + y < 2n$ muss gelten, $x = y = 0$ oder $x + y = n$. Dies ist aber im Widerspruch zu $y \neq x, y \neq n - x$. Wenn $n \mid (x - y)$ gilt, ergibt sich ebenfall $x = y$, was im Widerspruch steht zur Annahme. Daher folgert man:

$$p \mid (x + y), q \mid (x - y) \qquad \text{oder} \qquad p \mid (x - y), q \mid (x + y).$$

   Das wiederum impliziert:

$$\mathrm{ggT}(x + y, n) = p \text{ oder } q.$$

Da a vier Quadratwurzeln modulo n hat, ist die Erfolgswahrscheinlichkeit für jeden versuch $1/2$. Die erwartete Anzahl von Versuchen, bis ein Faktor von n gefunden ist, ist 2. Daher erwartet man in polynomialer Zeit das Egebnis.

Kapitel 18

Bestimmung des ggT zweier Zahlen

In diesem Abschnitt sehen wir uns explizit drei Algorithmen an, die in der Kryptologie eine sehr wichtige Rolle spielen.

18.1 Der Euklidische Algorithmus

Eine der wichtigsten Techniken in der Zahlentheorie ist der EUKLIDische Algorithmus, der ein einfaches aber sehr effektives Verfahren darstellt, um den größten gemeinsamen Teiler zweier Zahlen zu bestimmen. Eine erweiterte Form dieses Algorithmus ist der **Erweiterter Euklidischer Algorithmus**. Diese bestimmt ebenfalls den ggT zweier Zahlen und — falls der ggT der Zahlen den Wert 1 hat, diese also coprim sind — das multiplikative Inverse einer Zahl modulo der anderen Zahl.

Der EUKLIDische Algorithmus basiert auf dem folgenden Theorem:

Theorem [13]:

Für jede nichtnegative Zahl a und jede positive Zahl b gilt:

$$\mathrm{ggT}(a,b) = \mathrm{ggT}(b, a \bmod b)$$

Beweis:

Wir zeigen die Aussage in zwei Schritten.

① Wenn eine Zahl d gemeinsamer Teiler der beiden Zahlen a und b ist, dann

ist d auch gemeinsamer Teiler der Zahlen b und $a \bmod b$.

② Wenn d größter Teiler von a und b ist, dann ist d dies auch für die beiden Zahlen b und $a \bmod b$.

Zu ① Wir setzen:

$$d = \mathtt{ggT}(a, b).$$

Gemäß Definition des größten gemeinsamen Teilers bedeutet dies, dass d die Zahlen a und b teilt.

$$d \mid a \text{ und } d \mid b.$$

Für eine beliebige ganze positive Zahl b können wir a in der Form schreiben:

$$a = kb + r \equiv r \bmod b,$$

mit anderen Worten, wenn a durch b geteilt wird, bleibt Rest r, *i.e.*

$$r = a \bmod b$$

oder

$$a - kb = a \bmod b$$

für eine Zahl k. Da $d \mid b$ gilt, ist d auch Teiler von kb. Gemäß Annahme gilt aucht $d \mid a$. Daher können wir schließen, dass $d \mid (a - kb)$, dies wiederum impliziert $d \mid (a \bmod b)$.

Damit ist gezeigt, wenn d der größte gemeinsame Teiler von a und b ist, dann ist d auch gemeinsamer Teiler von b und $a \bmod b$. Wir haben noch nicht gezeigt, dass d auch der *größte* gemeinsame Teiler ist.

Zu ② Um zu zeigen, dass d auch der *größte* gemeinsame Teiler von b und $a \bmod b$ ist, zeigen wir, dass die Menge der gemeinsamen Teiler von a und b und b und $a \bmod b$ gleich sind. Wenn dies gezeigt ist, dann können wir schließen, daß d Conversely, we must show, if d is a common divisor of b and $a \bmod b$ then it divides a and b.

So, let d be a common divisor of b and $a \bmod b$. It follows, if $d \mid b$ then we also have $d \mid k \cdot b$. But then $d \mid (kb + (a \bmod b))$. This is the same as $d \mid a$.

Daher ist die Menge der Teiler von a und b die gleiche wie die der gemeinsamen Teiler von b und $a \bmod b$. Daher ist der größte gemeinsame Teiler der einen Menge auch gleich dem ggT der anderen Menge.

Die im obigen Theorem bewiesene Gleichung

$$\mathtt{ggT}(a, b) = \mathtt{ggT}(b, a \bmod b)$$

kann iterativ genutzt werden, um den größten gemeinsamen Teiler zweier Zahlen zu finden.

Beispiel [18.43]

Sei beispielsweise $a = 18, b = 12$, dann:

$$\begin{aligned} \mathtt{ggT}(18,12) &= \mathtt{ggT}(12,6) \\ &= \mathtt{ggT}(6,0) \\ &= 6 \end{aligned}$$

Betrachte $a = 21, b = 20$ dann:

$$\begin{aligned} \mathtt{ggT}(21,20) &= \mathtt{ggT}(20,1) \\ &= \mathtt{ggT}(1,0) \\ &= 1 \end{aligned}$$

Wir präsentieren hier den EUKLIDischen Algorithmus in mehreren Versionen, die alle zueinander äquivalent sind.

❶ **Version 1**

Wir zeigen den EUKLIDischen Algorithmus zunächst an einem konkreten Beispiel und geben anschließend das allgemeine Verfahren in der ersten Version an. Sei $a_0 = 174$ und $a_1 = 102$. Gesucht ist nun $\mathtt{ggT}(174, 102)$.

Step 1: Zunächst wird 174 ganzzahlig durch 102 geteilt:

$$174 = 1 * 102 + 72$$

Step 2: Anschließend wird 102 ganzzahlig durch den Rest 72 geteilt:

$$102 = 1 * 72 + 30$$

Step 3: Anschließend wird 72 ganzzahlig durch den Rest 30 geteilt:

$$72 = 2 * 30 + 12$$

Step 4: Anschließend wird 30 ganzzahlig durch den Rest 12 geteilt:

$$30 = 2 * 12 + 6$$

Step 5: Anschließend wird 12 ganzzahlig durch den Rest 6 geteilt:

$$12 = 2 * 6 + 0$$

Bei der letzten Division ist der Rest 0, der Algorithmus terminiert. Der letzte von 0 verschiedene Rest ist der gesucht größte gemeinsame Teiler, *i.e.*

$$\texttt{ggT}(174, 102) = 6.$$

Um eine möglichst einheitliche Notation verwenden zu können, nennen wir die beiden Ausgangszahlen a_0 und a_1 und fordern, dass stets $a_0 \geq a_1 > 0$ ist. Im folgenden ist die elementare Version des EUKLIDischen Algorithmus dargestellt:

Euklidischer Algorithmus Version 1:

E0 Input: $a_0, a_1 \in \mathbb{N}, a_0 \geq a_1 > 0$

E1 Falls $a_0 = a_1$ STOP $\texttt{ggT}(a_0, a_1) = a_0$.

E2 Sonst: Teile mit Rest:

$$\begin{aligned}
[0] \quad & a_0 = q_1 a_1 + a_2 \quad 0 < a_2 < a_1 \\
[1] \quad & a_1 = q_2 a_2 + a_3 \quad 0 < a_3 < a_2 \\
[2] \quad & a_2 = q_3 a_3 + a_4 \quad 0 < a_4 < a_3 \\
& \vdots \qquad \vdots \\
[k-3] \quad & a_{k-3} = q_{k-2} a_{k-2} + a_{k-1} \quad 0 < a_{k-1} < a_{k-2} \\
[k-2] \quad & a_{k-2} = q_{k-1} a_{k-1} + a_k \quad 0 < a_k < a_{k-1} \\
[k-1] \quad & a_{k-1} = q_k a_k + 0
\end{aligned}$$

bis zum ersten Mal der Rest 0 ist.

E3 STOP $\texttt{ggT}(a_0, a_1) = a_k$.

Beispiel [18.44]

Wie berechnen den $\texttt{ggT}(377, 233)$

Schritt 1: Input: Die natürlichen Zahlen $a_0 = 377$ und $a_1 = 233$

Schritt 2: Test: Ist $a_0 = a_1$? Nein, weiter mit **E2**:

Schritt 3: Teilen mit Rest:

$$\begin{aligned}
377 &= 1 \times 233 + 144 \\
233 &= 1 \times 144 + 89 \\
144 &= 1 \times 89 + 55 \\
89 &= 1 \times 55 + 34 \\
55 &= 1 \times 34 + 21 \\
34 &= 1 \times 21 + 13 \\
21 &= 1 \times 13 + 8 \\
13 &= 1 \times 8 + 5 \\
8 &= 1 \times 5 + 3 \\
5 &= 1 \times 3 + 2 \\
3 &= 1 \times 2 + 1 \\
2 &= 1 \times 1 + 1 \\
1 &= 1 \times 1 + 0
\end{aligned}$$

Schritt 4: STOP

$$\mathrm{ggT}(a_0, a_1) = \mathrm{ggT}(377, 233) = 1$$

Übung:

Berechnen Sie mit Hilfe des EUKLIDischen Algorithmus

(a) $\mathrm{ggT}(100, 35)$

(b) $\mathrm{ggT}(89, 55)$

(c) $\mathrm{ggT}(76084, 63020)$

Wir zeigen nun die Korrektheit des EUKLIDischen Algorithmus in dem folgenden Satz.

Satz:

Der EUKLIDische Algorithmus liefert in jedem Fall den größten gemeinsamen Teiler der Zahlen a_0 und a_1, i.e. $\mathrm{ggT}(a_0, a_1)$.

Beweis:

Aus Schritt $[k-1]$ folgt, daß a_k Teiler von a_{k-1} ist. Da a_k ein Teiler von a_{k-1} und natürlich auch ein Teiler von sich selbst ist, folgt aus Step $[k-2]$, daß a_k auch ein Teiler von a_{k-2} ist. Da a_k Teiler von a_{k-2} ist und Teiler von a_{k-1}, muß gemäß Step $[k-3]$ a_k auch Teiler von a_{k-3} sein. Diese Argumentationskette wird fortgesetzt bis zum ersten Schritt

[0], was implementiert, daß a_k Teiler von a_0 und Teiler von a_1 ist. Dies impliziert, daß a_k gemeinsamer Teiler der beiden Ausgangszahlen a_0 und a_1 ist. Es bleibt zu zeigen, daß a_k auch der größte gemeinsame Teiler ist.

Sei daher t irgendein anderer gemeinsamer Teiler von a_0 und a_1. Dann folgt aus [0], daß t auch Teiler von a_2 sein muß, denn die Terme a_0 und $q_1 \times a_1$ werden ja von t geteilt. Aus [1] folgt dann, daß t auch ein Teiler von a_3 sein muß. Aufgrund des gleichen Arguments muß wegen [2] t auch ein Teiler von a_4 sein. Diese Argumentationskette geht bis Schritt $[k-1]$. Dieser letzte Schritt impliziert, daß t auch ein Teiler von a_k sein muß. Daraus folgt, daß $t \leq a_k$, denn jeder Teiler einer Zahl ist höchstens so groß wie die Zahl selbst. Da t ein *beliebiger* gemeinsamer Teiler von a_0 und a_1 ist und $t \leq a_k$ haben wir gezeigt, daß a_k der größte gemeinsame Teiler der beiden zahlen a_0 und a_1 ist.

Eine wichtige Konsequenz aus dem EUKLIDischen Algorithmus ist folgender Satz:

Satz:

Sind $a_0 \geq 0$ und $a_1 \geq 0$ zwei natürliche Zahlen, so existieren $x, y \in \mathbb{Z}$ mit

$$\mathbf{ggT}(a_0, a_1) = x \cdot a_0 + y \cdot a_1$$

Beweis:

Um diese Behauptung zu beweisen, löst man das Gleichungssystem

$$\begin{array}{lll}
[0] & a_0 = q_1 \cdot a_1 + a_2 & \text{mit } 0 < a_2 < a_1 \\
[1] & a_1 = q_2 \cdot a_2 + a_3 & \text{mit } 0 < a_3 < a_2 \\
[2] & a_2 = q_3 \cdot a_3 + a_4 & \text{mit } 0 < a_4 < a_3 \\
 & \vdots \quad \vdots & \\
[k-3] & a_{k-3} = q_{k-2} \cdot a_{k-2} + a_{k-1} & \text{mit } 0 < a_{k-1} < a_{k-2} \\
[k-2] & a_{k-2} = q_{k-1} \cdot a_{k-1} + a_k & \text{mit } 0 < a_k < a_{k-1} \\
[k-1] & a_{k-1} = q_k \cdot a_k + 0 &
\end{array}$$

sukzessive nach a_0 und a_1 auf. Aus [0] folgt:

$$a_2 = a_0 - q_1 \times a_1$$

Aus [1] folgt:

$$\begin{aligned}
a_3 &= a_1 - q_2 \times a_2 \\
&= a_1 - q_2 \times (a_0 - q_1 \times a_1) \\
&= (-q_2)a_0 + (1 + q_1 q_2)a_1
\end{aligned}$$

Aus [2] erhält man:

$$\begin{aligned} a_4 &= a_2 - q_3 \times a_3 \\ &= a_0 - q_1 \times a_1 - q_3 \times \big((-q_2)a_0 + (1 + q_1 q_2)a_1\big) \\ &= (1 + q_2 q_3)a_0 - (q_1 + q_3 + q_1 q_2 q_3)a_1 \end{aligned}$$

Auf diese Weise fährt man bis Gleichung $[k-1]$ fort und schreibt letztendlich a_k als Linearkombinateion von a_0 und a_1. Damit verfügt man über ein konstruktives Verfahren, die Faktoren x und y der Linearkombination zu berechnen.

Beispiel:

Wir berechnen in diesem Beispiel explizit die Koeffizienten x und y für $a_0 = 100, a_1 = 35$.

Aus

$$a_0 = q_1 a_1 + a_2$$

folgt

$$100 = 2 \cdot 35 + 30$$

oder

$$30 = 100 - 2 \cdot 35$$

Aus

$$a_1 = q_2 \cdot a_2 + a_3$$

erält man

$$35 = 1 \cdot 30 + 5$$

oder

$$\begin{aligned} 5 &= 35 - 1 \cdot 30 \\ &= 35 - 1 \cdot (100 - 2 \cdot 35) \\ &= (-1) \cdot 100 + 3 \cdot 35 \end{aligned}$$

Die dritte Iteration ergibt

$$a_2 = q_3 \cdot a_3 + a_4$$

oder

$$30 = 6 \cdot 5 + 0$$

d.h. $a_3 = 5$ ist der **ggT** von 100 und 35 und aus dem verherigen Schritt erhält man:

$$\begin{aligned} 5 &= (-1) \cdot 100 + 3 \cdot 35 \\ &= x \cdot a_0 + y \cdot a_1 \end{aligned}$$

was auf $x = -1$ und $y = 3$ führt.

❷ Version 2

Im folgenden bezeichnen wir den größten gemeinsamen Teiler zweier Zahlen $n, m \in \mathbb{N}$ mit $\mathtt{ggT}(m, n)$.

Die aus der Schulmathematik bekannte Möglichkeit, die Zahl $\mathtt{ggT}(n, m)$ zu bestimmen, besteht in der Primfaktorzerlegung von n und m $(n, m \neq 0)$.

Alternativ kann auch der EUKLIDische Algorithmus in der iterativen Form

(a) $n = m \quad \longrightarrow \quad \mathtt{ggT}(n, m) = n \quad$ oder m

(b) $n < m \quad \longrightarrow \quad \mathtt{ggT}(n, m) = \mathtt{ggT}(m, n)$

(c) $n > m \quad \longrightarrow \quad \mathtt{ggT}(n, m) = \mathtt{ggT}(n - m, m)$

verwendet werden.

Beispiele:

(a) $\mathbf{ggT}(9, 6)$
Die Primfaktorenzerlegung liefert:

$$\begin{aligned} 9 &= 3 \times 3 \\ 6 &= 2 \times 3 \end{aligned}$$

woraus folgt: $\mathbf{ggT}(9, 6) = 3$.

Mit Hilfe des EUKLIDischen Algorithmus erhält man:

$$\begin{aligned} \mathtt{ggT}(9, 6) &\overset{(3)}{=} \mathtt{ggT}(3, 6) \\ &\overset{(2)}{=} \mathtt{ggT}(6, 3) \\ &\overset{(3)}{=} \mathtt{ggT}(3, 3) \\ &\overset{(1)}{=} 3 \end{aligned}$$

(b) $\mathtt{ggT}(72, 52)$
Die Primfaktorzerlegung liefert für dieses Beispiel:

$$\begin{aligned} 72 &= 2 \times 36 \\ &= 2 \times 2 \times 18 \\ &= 2 \times 2 \times 2 \times 9 \\ &= 2 \times 2 \times 2 \times 3 \times 3 \end{aligned}$$

und

$$\begin{aligned} 52 &= 2 \times 26 \\ &= 2 \times 2 \times 13 \end{aligned}$$

woraus folgt: $\mathtt{ggT}(72, 52) = 4$.

Mit Hilfe des EUKLIDischen Algorithmus erhält man hier:

$$\begin{aligned}
\mathrm{ggT}(72,52) &\overset{(3)}{=} \mathrm{ggT}(20,52)\\
&\overset{(2)}{=} \mathrm{ggT}(52,20)\\
&\overset{(3)}{=} \mathrm{ggT}(32,20)\\
&\overset{(3)}{=} \mathrm{ggT}(12,20)\\
&\overset{(2)}{=} \mathrm{ggT}(20,12)\\
&\overset{(3)}{=} \mathrm{ggT}(8,12)\\
&\overset{(2)}{=} \mathrm{ggT}(12,8)\\
&\overset{(3)}{=} \mathrm{ggT}(4,8)\\
&\overset{(2)}{=} \mathrm{ggT}(8,4)\\
&\overset{(3)}{=} \mathrm{ggT}(4,4)\\
&\overset{(1)}{=} 4
\end{aligned}$$

Dies sind zwei explizite Beispiele dafür, wie der EUKLIDische Algorithmus arbeitet.

Damit lautet der Algorithmus in der Umgangssprache:

> Solange x von y verschieden ist, wird geprüft, ob x größer als y ist. Ist dies der Fall wird x durch $x - y$ ersetzt (Regel 3), andernfalls werden x und y vertauscht (Regel 2). Unterscheiden sich x und y nicht mehr, wird Regel 1 angewendet und der Rückgabewert der Funktion ist x.

❸ Version 3

Wir formulieren den EUKLIDischen Algorithmus nochmals aber diesmal in einer etwas formaleren Weise:

Der EUKLIDische Algorithmus macht wiederholten Gebrauch des obigen Satzes, um den größten gemeinsamen Teiler zu finden. Es wird angenommen, dass $d > f > 0$.

EUKLIDischer Algorithmus
EUCLID(d,f)

E0 Initialisiere $X \leftarrow d; Y \leftarrow f$.

E1 wenn $Y = 0$ return $X = \texttt{ggT}(d, f)$ `STOP` sonst gehe zu Schritt **E2**.

E2 berechne $R = X \bmod Y$

E3 ersetze $X \leftarrow Y$

E4 ersetze $Y \leftarrow R$; weiter mit Schritt **E1**.

Beispiel [18.45]

Wir berechnen mit Hilfe des EUKLIDischen Algorithmus den ggT von $d = 1970$ und $f = 1066$.

STEP 1 E0: Initialiisere $X = 1970, Y = 1066$

STEP 2 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 3 E2: berechne $R = X \bmod Y = 1970 \bmod 1066 = 904$

STEP 4 E3, E4: $X = 1066$ und $Y = 904$; weiter mit **E1**.

STEP 5 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 6 E2: berechne $R = X \bmod Y = 1066 \bmod 904 = 162$

STEP 7 E3, E4: $X = 904$ und $Y = 162$; weiter mit **E1**.

STEP 8 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 9 E2: berechne $R = X \bmod Y = 904 \bmod 162 = 94$

STEP 10 E3, E4: $X = 162$ und $Y = 94$; weiter mit **E1**.

STEP 11 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 12 E2: berechne $R = X \bmod Y = 162 \bmod 94 = 68$

STEP 13 E3, E4: $X = 94$ und $Y = 68$; weiter mit **E1**.

STEP 14 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 15 E2: berechne $R = X \bmod Y = 94 \bmod 68 = 26$

STEP 16 E3, E4: $X = 68$ und $Y = 26$; weiter mit **E1**.

STEP 17 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 18 E2: berechne $R = X \bmod Y = 68 \bmod 26 = 16$

STEP 19 E3, E4: $X = 26$ und $Y = 16$; weiter mit **E1**.

STEP 20 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 21 E2: berechne $R = X \bmod Y = 26 \bmod 16 = 10$

STEP 22 E3, E4: $X = 16$ und $Y = 10$; weiter mit **E1**.

STEP 23 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 24 E2: berechne $R = X \bmod Y = 16 \bmod 10 = 6$

STEP 25 E3, E4: $X = 10$ und $Y = 6$; weiter mit **E1**.

STEP 26 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 27 E2: berechne $R = X \bmod Y = 10 \bmod 6 = 4$

STEP 28 E3, E4: $X = 6$ und $Y = 4$; weiter mit **E1**.

STEP 29 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 30 E2: berechne $R = X \bmod Y = 6 \bmod 4 = 2$

STEP 31 E3, E4: $X = 4$ und $Y = 2$; weiter mit **E1**.

STEP 32 E1: $Y = 0$? $\rightarrow$ nein; weiter mit **E2**.

STEP 30 E2: berechne $R = X \bmod Y = 4 \bmod 2 = 0$

STEP 31 E3, E4: $X = 2$ und $Y = 0$; weiter mit **E1**.

STEP 32 E1: $Y = 0$? $\rightarrow$ ja; return $X = \gcd(1970, 1066) = 2$.

Daher ist $\mathtt{ggT}(1970, 1066) = 2$.

18.2 Der erweiterte Euklidische Algorithmus

Der im vorigen Abschnitt untersuchte EUKLIDische Algorithmus kann erweitert werden, um das multiplikative Inverse zu berechnen.

Wir demonstrieren das Verfahren zunächst an einem Beispiel.

Beispiel [18.46]

Wir berechnen das multiplikative Inverse von 510 mod1001, *i.e.* gesucht ist die Zahl $x \in \mathbb{Z}_{1001}$ mit

$$510 \cdot x \equiv 1 \bmod 1001.$$

Wir berechnen dazu den $\mathtt{ggT}(1001, 510)$ mittels Divisionsverfahren:

Schritt 1: Dividiere 1001 durch 510

$$1001 = 1 \cdot 510 + 491.$$

Wir können nun die (ganzzahligen) Reste der Division durch die beiden Zahlen 1001 und 510 ausdrücken, es ist dann

$$491 = 1 \cdot 1001 - 1 \cdot 510. \tag{18.1}$$

Schritt 2: Dividiere 510 durch 491

$$510 = 1 \cdot 491 + 19.$$

Wir drücken wieder den Rest 19 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 19 &= 1 \cdot 510 - 1 \cdot 491 \\ &\overset{(18.6)}{=} 1 \cdot 510 - 1(1 \cdot 1001 - 1 \cdot 510) \\ &= 2 \cdot 510 - 1001. \end{aligned} \tag{18.2}$$

Schritt 3: Dividiere 491 durch 19

$$491 = 25 \cdot 19 + 16.$$

Wir drücken wieder den Rest 16 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 16 &= 491 - 25 \cdot 19 \\ &\overset{(18.6,18.7)}{=} 1 \cdot 1001 - 1 \cdot 510 - 25(2dot510 - 1001) \\ &= 26 \cdot 1001 - 51 \cdot 510. \end{aligned} \tag{18.3}$$

Schritt 4: Dividiere 19 durch 16:

$$19 = 1 \cdot 16 + 3.$$

Wir drücken wieder den Rest 3 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 3 &= 19 - 1 \cdot 16 \\ &\overset{(18.7,18.8)}{=} 2 \cdot 510 - 1001 - 26 \cdot 1001 + 51 \cdot 510 \\ &= 53 \cdot 510 - 27 \cdot 1001. \end{aligned} \tag{18.4}$$

Schritt 5: Dividiere 16 durch 3:

$$16 = 5 \cdot 3 + 1.$$

Wir drücken den Rest 1 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 1 &= 16 - 5 \cdot 3 \\ &\overset{(18.8,18.9)}{=} 26 \cdot 1001 - 51 \cdot 510 - 5(53 \cdot 510 - 27 \cdot 1001) \\ &= 161 \cdot 1001 - 316 \cdot 510. \end{aligned} \tag{18.5}$$

Das Divisionsverfahren endet an dieser Stelle. Wir haben daher:

$$1 = 161 \cdot 1001 - 316 \cdot 510.$$

Dies können wir auch schreiben in der Form:

$$-316 \cdot 510 = -161 \cdot 1001 + 1.$$

Da $-161 \in \mathbb{Z}$ erhalten wir:

$$-316 \cdot 510 \equiv 1 \bmod 1001.$$

Damit haben wir das multiplikative Inverse

$$510^{-1} \bmod 1001$$

berechnet. Es ist

$$510^{-1} \bmod 1001 = -316 \bmod 1001 \equiv 685 \bmod 1001.$$

Check:

$$510 \cdot 685 = 349\,350 = 349 \cdot 1001 + 1.$$

18.2.1 Der erweiterte Euklidische Algorithmus

Der EUKLIDische Algorithmus, den wir im letzten Abschnitt untersucht haben, kann erweitert werden, um das multiplikative Inverse zu bestimmen. Sind die beiden Zahlen d und f coprim, *i.e.* $\mathtt{ggT}(d, f) = 1$, dann hat d ein multiplikatives

Inverses modulo f. Das bedeutet also, für positive ganze Zahlen $d < f$, gibt es ein $d^{-1} < f$ so dass $d \cdot d^{-1} \equiv 1 \bmod f$.

Wir demonstrieren das Verfahren zunächst an einem Beispiel.

Beispiel [18.47]

Wir berechnen das multiplikative Inverse von 510 mod1001, *i.e.* gesucht ist die Zahl $x \in \mathbb{Z}_{1001}$ mit

$$510 \cdot x \equiv 1 \bmod 1001.$$

Wir berechnen dazu den $\texttt{ggT}(1001, 510)$:

Schritt 1: Dividiere 1001 durch 510

$$1001 = 1 \cdot 510 + 491.$$

Wir können nun die (ganzzahligen) Reste der Division durch die beiden Zahlen 1001 und 510 ausdrücken, es ist dann

$$491 = 1 \cdot 1001 - 1 \cdot 510. \tag{18.6}$$

Schritt 2: Dividiere 510 durch 491

$$510 = 1 \cdot 491 + 19.$$

Wir drücken wieder den Rest 19 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 19 &= 1 \cdot 510 - 1 \cdot 491 \\ &\overset{(18.6)}{=} 1 \cdot 510 - 1(1 \cdot 1001 - 1 \cdot 510) \\ &= 2 \cdot 510 - 1001. \end{aligned} \tag{18.7}$$

Schritt 3: Dividiere 491 durch 19

$$491 = 25 \cdot 19 + 16.$$

Wir drücken wieder den Rest 16 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 16 &= 491 - 25 \cdot 19 \\ &\overset{(18.6,18.7)}{=} 1 \cdot 1001 - 1 \cdot 510 - 25(2dot510 - 1001) \\ &= 26 \cdot 1001 - 51 \cdot 510. \end{aligned} \tag{18.8}$$

Schritt 4: Dividiere 19 durch 16:

$$19 = 1 \cdot 16 + 3.$$

Wir drücken wieder den Rest 3 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 3 &= 19 - 1 \cdot 16 \\ &\overset{(18.7,18.8)}{=} 2 \cdot 510 - 1001 - 26 \cdot 1001 + 51 \cdot 510 \\ &= 53 \cdot 510 - 27 \cdot 1001. \end{aligned} \tag{18.9}$$

Schritt 5: Dividiere 16 durch 3:

$$16 = 5 \cdot 3 + 1.$$

Wir drücken den Rest 1 als Linearkombonation von 1001 und 510 aus:

$$\begin{aligned} 1 &= 16 - 5 \cdot 3 \\ &\overset{(18.8,18.9)}{=} 26 \cdot 1001 - 51 \cdot 510 - 5(53 \cdot 510 - 27 \cdot 1001) \\ &= 161 \cdot 1001 - 316 \cdot 510. \end{aligned} \tag{18.10}$$

Das Divisionsverfahren endet an dieser Stelle. Wir haben daher:

$$1 = 161 \cdot 1001 - 316 \cdot 510.$$

Dies können wir auch schreiben in der Form:

$$-316 \cdot 510 = -161 \cdot 1001 + 1.$$

Da $-161 \in \mathbb{Z}$ erhalten wir:

$$-316 \cdot 510 \equiv 1 \bmod 1001.$$

Damit haben wir das multiplikative Inverse

$$510^{-1} \bmod 1001$$

berechnet. Es ist

$$510^{-1} \bmod 1001 = -316 \bmod 1001 \equiv 685 \bmod 1001.$$

Check:

$$510 \cdot 685 = 349\,350 = 349 \cdot 1001 + 1.$$

Der erweiterte EUKLIDische Algorithmus ist in [125] oder [52, p. 79] dargestellt.

Erweiterter Euklidischer Algorithmus

EXTENDED EUCLID(d,f)

ExE0 Initialisierung

$$(X_1, X_2, X_3) \leftarrow (1, 0, f); (Y_1, Y_2, Y_3) \leftarrow (0, 1, d).$$

ExE1 wenn $Y_3 = 0$ return $X_3 = \gcd(d, f)$ `STOP` kein Inverses.

ExE2 wenn $Y_3 = 1$ return $Y_3 = \gcd(d, f)$, $Y_2 = d^{-1} \bmod f$ `STOP`.

ExE3 berechne $Q = \left\lfloor \frac{X_3}{Y_3} \right\rfloor$.

ExE4 ersetze

$$\begin{aligned} T_1 &\longleftarrow X_1 - Q \cdot Y_1 \\ T_2 &\longleftarrow X_2 - Q \cdot Y_2 \\ T_3 &\longleftarrow X_3 - Q \cdot Y_3 \end{aligned}$$

ExE5 ersetze $(X_1, X_2, X_3) \leftarrow (Y_1, Y_2, Y_3)$.

ExE6 ersetze $(Y_1, Y_2, Y_3) \leftarrow (T_1, T_2, T_3)$.

ExE7 weiter mit Schritt **ExE1**.

Während der Berechnung gelten folgende Beziehungen

$$\begin{aligned} f \cdot T_1 + d \cdot T_2 &= T_3 \\ f \cdot X_1 + d \cdot X_2 &= X_3 \\ f \cdot Y_1 + d \cdot Y_2 &= Y_3 \end{aligned}$$

Anmerkung:

Dieser Algorithmus liefert — sofern die beiden Zahlen d, f coprim sind — die *beiden* multiplikativen Inversen. Das heißt,

$$Y_2 = d^{-1} \bmod f$$

und

$$Y_1 = f^{-1} \bmod d$$

Beispiel [**18.48**]

Wir spielen den Fall $f = 1769$ und $d = 550$ einmal durch. Gesucht ist also eine

Zahl d^{-1} mit der Eigenschft:

$$550 \cdot d^{-1} \equiv 1 \bmod 1769$$

STEP 1 ExE0: Initialisiere $(X_1, X_2, X_3) \leftarrow (1, 0, 1769)$, und $(Y_1, Y_2, Y_3) \leftarrow (0, 1, 550)$. Wir haben also als Startwerte $X_1 = 1, X_2 = 0, X_3 = 1769, Y_1 = 0, Y_2 = 1, Y_3 = 550$.

STEP 2 ExE1: $Y_3 = 0$? nein

STEP 3 ExE2: $Y_3 = 1$? nein

STEP 4 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{1769}{550} \rfloor = 3$.

STEP 5 ExE4: Berechne

$$\begin{aligned} T_1 &= X_1 - QY_1 = 1 - 3 \cdot 0 = 1 \\ T_2 &= X_2 - QY_2 = 0 - 3 \cdot 1 = -3 \\ T_3 &= X_3 - QY_3 = 1769 - 3 \cdot 550 = 119 \end{aligned}$$

STEP 6 ExE5, **ExE6** und **ExE7**: Ersetze

$$\begin{aligned} X_1 &\leftarrow Y_1 = 0 \\ X_2 &\leftarrow Y_2 = 1 \\ X_3 &\leftarrow Y_3 = 550 \end{aligned}$$

und

$$\begin{aligned} Y_1 &\leftarrow T_1 = 1 \\ Y_2 &\leftarrow T_2 = -3 \\ Y_3 &\leftarrow T_3 = 119 \end{aligned}$$

Damit haben wir: $X_1 = 0, X_2 = 1, X_3 = 550, Y_1 = 1, Y_2 = -3, Y_3 = 119$. Weiter mit **ExE1**.

STEP 7 ExE1: $Y_3 = 0$? nein

STEP 8 ExE2: $Y_3 = 1$? nein

STEP 9 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{550}{119} \rfloor = 4$.

STEP 10 ExE4: Berechnee

$$\begin{aligned} T_1 &= X_1 - QY_1 = 0 - 4 \cdot 1 = -4 \\ T_2 &= X_2 - QY_2 = 1 - 4 \cdot (-3) = 13 \\ T_3 &= X_3 - QY_3 = 550 - 4 \cdot 119 = 74 \end{aligned}$$

STEP 11 ExE5, **ExE6** und **ExE7**: Ersetze

$$\begin{aligned} X_1 &\leftarrow Y_1 = 1 \\ X_2 &\leftarrow Y_2 = -3 \\ X_3 &\leftarrow Y_3 = 119 \end{aligned}$$

und

$$Y_1 \leftarrow T_1 = -4$$
$$Y_2 \leftarrow T_2 = 13$$
$$Y_3 \leftarrow T_3 = 74$$

Damit haben wir nach dieser Runde die Werte: $X_1 = 1, X_2 = -3, X_3 = 119, Y_1 = -4, Y_2 = 13, Y_3 = 74$. Weiter mit **ExE1**.

STEP 12 ExE1: $Y_3 = 0$? nein

STEP 13 ExE2: $Y_3 = 1$? nein

STEP 14 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{119}{74} \rfloor = 1$.

STEP 15 ExE4: Berechne

$$T_1 = X_1 - QY_1 = 1 - 1 \cdot (-4) = 5$$
$$T_2 = X_2 - QY_2 = -3 - 1 \cdot 13 = -16$$
$$T_3 = X_3 - QY_3 = 119 - 1 \cdot 74 = 45$$

STEP 16 ExE5, **ExE6** und **ExE7**: Ersetze

$$X_1 \leftarrow Y_1 = -4$$
$$X_2 \leftarrow Y_2 = 13$$
$$X_3 \leftarrow Y_3 = 74$$

und

$$Y_1 \leftarrow T_1 = 5$$
$$Y_2 \leftarrow T_2 = -16$$
$$Y_3 \leftarrow T_3 = 45$$

Damit haben wir: $X_1 = -4, X_2 = 13, X_3 = 74, Y_1 = 5, Y_2 = -16, Y_3 = 45$. Weiter mit **ExE1**.

STEP 17 ExE1: $Y_3 = 0$? nein

STEP 18 ExE2: $Y_3 = 1$? nein

STEP 19 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{74}{45} \rfloor = 1$.

STEP 20 ExE4: Berechne

$$T_1 = X_1 - QY_1 = -4 - 1 \cdot 5 = -9$$
$$T_2 = X_2 - QY_2 = 13 - 1 \cdot (-16) = 29$$
$$T_3 = X_3 - QY_3 = 74 - 1 \cdot 45 = 29$$

STEP 21 ExE5, **ExE6** und **ExE7**: Ersetze

$$X_1 \leftarrow Y_1 = 5$$
$$X_2 \leftarrow Y_2 = -16$$
$$X_3 \leftarrow Y_3 = 45$$

und

$$Y_1 \leftarrow T_1 = -9$$
$$Y_2 \leftarrow T_2 = 29$$
$$Y_3 \leftarrow T_3 = 29$$

Damit: $X_1 = 5, X_2 = -16, X_3 = 45, Y_1 = -9, Y_2 = 29, Y_3 = 29$. Weiter mit **ExE1**.

STEP 22 ExE1: $Y_3 = 0$? no

STEP 23 ExE2: $Y_3 = 1$? no

STEP 24 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{45}{29} \rfloor = 1$.

STEP 25 ExE4: Berechne

$$T_1 = X_1 - QY_1 = 5 - 1 \cdot (-9) = 14$$
$$T_2 = X_2 - QY_2 = -16 - 1 \cdot 29 = -45$$
$$T_3 = X_3 - QY_3 = 45 - 1 \cdot 29 = 16$$

STEP 26 ExE5, **ExE6** and **ExE7**: Ersetze

$$X_1 \leftarrow Y_1 = -9$$
$$X_2 \leftarrow Y_2 = 29$$
$$X_3 \leftarrow Y_3 = 29$$

und

$$Y_1 \leftarrow T_1 = 14$$
$$Y_2 \leftarrow T_2 = -45$$
$$Y_3 \leftarrow T_3 = 16$$

Damit: $X_1 = -9, X_2 = 29, X_3 = 29, Y_1 = 14, Y_2 = -45, Y_3 = 16$. Weiter mit **ExE1**.

STEP 27 ExE1: $Y_3 = 0$? nein

STEP 28 ExE2: $Y_3 = 1$? nein

STEP 29 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{29}{16} \rfloor = 1$.

STEP 30 ExE4: Berechne

$$T_1 = X_1 - QY_1 = -9 - 1 \cdot 14 = -23$$
$$T_2 = X_2 - QY_2 = 29 - 1 \cdot (-45) = 74$$
$$T_3 = X_3 - QY_3 = 29 - 1 \cdot 16 = 13$$

STEP 31 ExE5, **ExE6** und **ExE7**: Ersetze

$$X_1 \leftarrow Y_1 = 14$$
$$X_2 \leftarrow Y_2 = -45$$
$$X_3 \leftarrow Y_3 = 16$$

und

$$Y_1 \leftarrow T_1 = -23$$
$$Y_2 \leftarrow T_2 = 74$$
$$Y_3 \leftarrow T_3 = 13$$

Damit: $X_1 = 14, X_2 = -45, X_3 = 16, Y_1 = -23, Y_2 = 74, Y_3 = 13$. Weiter mit **ExE1**.

STEP 32 ExE1: $Y_3 = 0$? nein

STEP 33 ExE2: $Y_3 = 1$? nein

STEP 34 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{16}{13} \rfloor = 1$.

STEP 35 ExE4: Berechne

$$T_1 = X1 - Q \cdot Y_1 = 14 - 1 \cdot (-23) = 37$$
$$T_2 = X2 - Q \cdot Y_2 = -45 - 1 \cdot 74 = -119$$
$$T_3 = X3 - Q \cdot Y_3 = 16 - 1 \cdot 13 = 3$$

STEP 36 ExE5, **ExE6** und **ExE7**: Ersetze

$$X_1 \leftarrow Y_1 = -23$$
$$X_2 \leftarrow Y_2 = 74$$
$$X_3 \leftarrow Y_3 = 13$$

und

$$Y_1 \leftarrow T_1 = 37$$
$$Y_2 \leftarrow T_2 = -119$$
$$Y_3 \leftarrow T_3 = 3$$

Damit erhalten wir: nach dieser Runde:

$$X_1 = -23, X_2 = 74, X_3 = 13, Y_1 = 37, Y_2 = -119, Y_3 = 3.$$

Weiter mit **ExE1**.

STEP 37 ExE1: $Y_3 = 0$? nein

STEP 38 ExE2: $Y_3 = 1$? nein

STEP 39 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{13}{3} \rfloor = 4$.

STEP 40 ExE4: Berechne

$$T_1 = X_1 - QY_1 = -23 - 4 \cdot 37 = -171$$
$$T_2 = X_2 - QY_2 = 74 - 4 \cdot (-119) = 550$$
$$T_3 = X_3 - QY_3 = 13 - 4 \cdot 3 = 1$$

STEP 41 ExE5, **ExE6** und **ExE7**: Ersetze

$$X_1 \leftarrow Y_1 = 37$$
$$X_2 \leftarrow Y_2 = -119$$
$$X_3 \leftarrow Y_3 = 3$$

und

$$Y_1 \leftarrow T_1 = -171$$
$$Y_2 \leftarrow T_2 = 550$$
$$Y_3 \leftarrow T_3 = 1$$

Damit erhalten wir also: $X_1 = 37, X_2 = -119, X_3 = 3, Y_1 = -171, Y_2 = 550, Y_3 = 1$. Weiter mit **ExE1**.

STEP 42 ExE1: $Y_3 = 0$? nein

STEP 43 ExE2: $Y_3 = 1$? ja
Damit ist $\texttt{ggT}(550, 1769) = 1$ und das multiplikative Inverse von 550 ist: $Y_2 = 550$, i.e.

$$550 \cdot 550 \equiv 1 \bmod 1769$$

Beispiel [18.49]

In einem zweiten Beispiel berechnen wir mit Hilfe des erweiterten EUKLIDischen Algorithmus das multiplikative Inverse von 11 mod 64.

STEP 1 ExE0: Initialisierung $(X_1, X_2, X_3) \leftarrow (1, 0, 64)$, und $(Y_1, Y_2, Y_3) \leftarrow (0, 1, 11)$. Wir haben daher als Anfangswerte: $X_1 = 1, X_2 = 0, X_3 = 64, Y_1 = 0, Y_2 = 1, Y_3 = 11$.

STEP 2 ExE1: $Y_3 = 0$? nein

STEP 3 ExE2: $Y_3 = 1$? nein

STEP 4 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{64}{11} \rfloor = 5$.

STEP 5 ExE4: Berechne

$$T_1 = X_1 - Q \cdot Y_1 = 1 - 5 \cdot 0 = 1$$
$$T_2 = X_2 - Q \cdot Y_2 = 0 - 5 \cdot 1 = -5$$
$$T_3 = X_3 - Q \cdot Y_3 = 64 - 5 \cdot 11 = 9$$

STEP 6 ExE5, **ExE6** und **ExE7**: ersetze

$$X_1 \leftarrow Y_1 = 0$$
$$X_2 \leftarrow Y_2 = 1$$
$$X_3 \leftarrow Y_3 = 11$$

und

$$Y_1 \leftarrow T_1 = 1$$
$$Y_2 \leftarrow T_2 = -5$$
$$Y_3 \leftarrow T_3 = 9$$

Daher haben wir nach dieser Runde: $X_1 = 0, X_2 = 1, X_3 = 11, Y_1 = 1, Y_2 = -5, Y_3 = 9$. Weiter mit Schritt **ExE1**.

STEP 7 ExE1: $Y_3 = 0$? nein

STEP 8 ExE2: $Y_3 = 1$? nein

STEP 9 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{11}{9} \rfloor = 1$.

STEP 10 ExE4: Berechne

$$T_1 = X_1 - Q \cdot Y_1 = 0 - 1 \cdot 1 = -1$$
$$T_2 = X_2 - Q \cdot Y_2 = 1 - 1 \cdot (-5) = 6$$
$$T_3 = X_3 - Q \cdot Y_3 = 11 - 1 \cdot 9 = 2$$

STEP 11 ExE5, **ExE6** und **ExE7**: ersetze

$$X_1 \leftarrow Y_1 = 1$$
$$X_2 \leftarrow Y_2 = -5$$
$$X_3 \leftarrow Y_3 = 9$$

und

$$Y_1 \leftarrow T_1 = -1$$
$$Y_2 \leftarrow T_2 = 6$$
$$Y_3 \leftarrow T_3 = 2$$

Damit haben wir: $X_1 = 1, X_2 = -5, X_3 = 9, Y_1 = -1, Y_2 = 6, Y_3 = 2$. Weiter mit Schritt **ExE1**.

STEP 12 ExE1: $Y_3 = 0$? nein

STEP 13 ExE2: $Y_3 = 1$? nein

STEP 14 ExE3: Berechne $Q = \lfloor \frac{X_3}{Y_3} \rfloor = \lfloor \frac{9}{2} \rfloor = 4$.

STEP 15 ExE4: Berechne

$$T_1 = X_1 - Q \cdot Y_1 = 1 - 4 \cdot (-1) = 5$$
$$T_2 = X_2 - Q \cdot Y_2 = -5 - 4 \cdot 6 = -29$$
$$T_3 = X_3 - Q \cdot Y_3 = 9 - 4 \cdot 2 = 1$$

STEP 16 ExE5, **ExE6** und **ExE7**: ersetze

$$X_1 \leftarrow Y_1 = 1$$
$$X_2 \leftarrow Y_2 = 6$$
$$X_3 \leftarrow Y_3 = 2$$

und

$$Y_1 \leftarrow T_1 = 5$$
$$Y_2 \leftarrow T_2 = -29$$
$$Y_3 \leftarrow T_3 = 1$$

Daher: $X_1 = 1, X_2 = 6, X_3 = 2, Y_1 = 5, Y_2 = -29, Y_3 = 1$. Weiter mit Schritt **ExE1**.

STEP 17 ExE1: $Y_3 = 0$? nein

STEP 18 ExE2: $Y_3 = 1$? ja
Damit ist $\mathrm{ggT}(11, 64) = 1$ und das multiplikative Inverse von 11 modulo 64 die Zahl -29, i.e.

$$11 \cdot (-29) \equiv 1 \bmod 64$$

Da -29 kongruent zu 35 modulo 64, können wir eine Zahl d^{-1} angeben mit $0 < d^{-1} < f$, so daß

$$d \cdot d^{-1} \equiv 1 \bmod f$$

denn

$$11 \cdot 35 = 385 = 384 + 1 = 6 \cdot 64 + 1 \equiv 1 \bmod 64$$

Bei genauer Untersuchung der Arbeitsweise des erweiterten EUKLIDischen Algorithmus fällt auf, dass wir bisher von den Variablen X_1 und Y_1 keinen Gebrauch gemacht haben. Da in dem Verfahren die X_1, Y_1 Variablen nicht mit den X_2, Y_2 Variablen gemischt werden, scheinen erste überflüssig. Die X_1, Y_1 Variablen sind dann von Interesse, wenn man das multiplikative Inverse von 64 modulo 11 ermitteln will.

Da

$$64 \bmod 11 \equiv 9 \bmod 11,$$

findet man mit dem Wert von $Y_1 = 5$ aus obiger Rechnung, dass in der Tat

$$9 \cdot 5 = 45 \equiv 1 \bmod 11.$$

18.3 Der Algorithmus von Josef Stein

Seit mehr als 2000 Jahren ist der EUKLIDische Algorithmus *das* Verfahren, um den größten gemeinsamen Teiler zweier Zahlen zu finden. Im Jahre 1967 wurde von JOSEF STEIN ein alternatives Verfahren entwickelt[1], das auf dem folgenden Algorithmus basiert (siehe [126], chapter 4.5.2.)

JOSEF STEIN Algorithmus:

Der folgende Algorithmus bestimmt den ggT der beiden Zahlen A, B:

Step 0 Input: Zwei Zahlen A, B.

Step 1 Initialisierung: Setze $A_1 \longleftarrow A, B_1 \longleftarrow B$ und $C_1 \longleftarrow 1$

Step n 1. Falls $A_n = B_n$ STOP $\quad \text{ggT}(A, B) = A_n \cdot C_n$.

2. Falls A_n und B_n beide gerade sind, setze:

$$\begin{aligned} A_{n+1} &\longleftarrow A_n/2 \\ B_{n+1} &\longleftarrow B_n/2 \\ C_{n+1} &\longleftarrow 2 \cdot C_n \end{aligned}$$

3. Wenn A_n gerade und B_n ungerade ist, setze:

$$\begin{aligned} A_{n+1} &\longleftarrow A_n/2 \\ B_{n+1} &\longleftarrow B_n \\ C_{n+1} &\longleftarrow C_n \end{aligned}$$

4. Wenn A_n ungerade und B_n gerade ist, setze:

$$\begin{aligned} A_{n+1} &\longleftarrow A_n \\ B_{n+1} &\longleftarrow B_n/2 \\ C_{n+1} &\longleftarrow C_n \end{aligned}$$

5. Falls beide Zahlen A_n und B_n ungerade sind, setze

$$\begin{aligned} A_{n+1} &\longleftarrow \mid A_n - B_n \mid \\ B_{n+1} &\longleftarrow \min\{A_n, B_n\} \\ C_{n+1} &\longleftarrow C_n \end{aligned}$$

Weiter mit **Step n + 1** .

[1] See J. Stein, J. Comp. Phys. **1**, (1967), 397 – 405.

Beispiel:

In diesem Beispiel bestimmen wir mit Hilfe des STEIN Algorithmus den ggT von $A = 2152$ and $B = 764$.

Step 1: Initialisierung: $A_1 = A = 2152; B_1 = B = 764; C_1 = 1$

Step 2: Da A_1 und B_1 beide gerade sind, setzen wir:

$$\begin{aligned} A_2 &\longleftarrow A_1/2 = 1076 \\ B_2 &\longleftarrow B_1/2 = 382 \\ C_2 &\longleftarrow 2 \cdot C_1 = 2 \end{aligned}$$

Step 3: Da A_2 und B_2 beide gerade sind, setzen wir:

$$\begin{aligned} A_3 &\longleftarrow A_1/2 = 538 \\ B_3 &\longleftarrow B_1/2 = 191 \\ C_3 &\longleftarrow 2 \cdot C_2 = 4 \end{aligned}$$

Step 4: Da A_3 gerade und B_3 ungerade ist, setzen wir:

$$\begin{aligned} A_4 &\longleftarrow A_3/2 = 269 \\ B_4 &\longleftarrow B_3 = 191 \\ C_4 &\longleftarrow C_3 = 4 \end{aligned}$$

Step 5: Da die beiden Zahlen A_4 und B_4 ungerade sind, folgt:

$$\begin{aligned} A_5 &\longleftarrow \mid A_4 - B_4 \mid = 78 \\ B_5 &\longleftarrow \min\{A_4, B_4\} = 191 \\ C_5 &\longleftarrow C_4 = 4 \end{aligned}$$

Step 6: Da A_5 gerade und B_5 ungerade ist, erhalten wir:

$$\begin{aligned} A_6 &\longleftarrow A_5/2 = 39 \\ B_6 &\longleftarrow B_5 = 191 \\ C_6 &\longleftarrow C_5 = 4 \end{aligned}$$

Step 7: A_6 und B_6 sind beide ungerade, wir setzen daher:

$$\begin{aligned} A_7 &\longleftarrow \mid A_6 - B_6 \mid = 152 \\ B_7 &\longleftarrow \min\{A_6, B_6\} = 39 \\ C_7 &\longleftarrow C_6 = 4 \end{aligned}$$

Step 8: A_7 ist gerade und B_7 ist ungerade, wir setzen daher

$$\begin{aligned} A_8 &\longleftarrow A_7/2 = 76 \\ B_8 &\longleftarrow B_7 = 39 \\ C_8 &\longleftarrow C_7 = 4 \end{aligned}$$

Step 9: Hier ist A_8 ebenfalls gerade und B_8 ungerade:

$$\begin{aligned} A_9 &\longleftarrow A_8/2 = 38 \\ B_9 &\longleftarrow B_8 = 39 \\ C_9 &\longleftarrow C_8 = 4 \end{aligned}$$

Step 10: A_9 ist gerade und B_9 ist ungerade, daher

$$\begin{aligned} A_{10} &\longleftarrow A_9/2 = 19 \\ B_{10} &\longleftarrow B_9 = 39 \\ C_{10} &\longleftarrow C_9 = 4 \end{aligned}$$

Step 11: Beide Zahlen sind ungerade:

$$\begin{aligned} A_{11} &\longleftarrow\mid A_{10} - B_{10} \mid = 20 \\ B_{11} &\longleftarrow \min\{A_{10}, B_{10}\} = 19 \\ C_{11} &\longleftarrow C_{10} = 4 \end{aligned}$$

Step 12: A_{11} ist gerade und B_{11} ist ungerade, also:

$$\begin{aligned} A_{12} &\longleftarrow A_{11}/2 = 10 \\ B_{12} &\longleftarrow B_{11} = 19 \\ C_{12} &\longleftarrow C_{11} = 4 \end{aligned}$$

Step 13: Da A_{12} gerade und B_{12} ungerade ist, setzen wir:

$$\begin{aligned} A_{13} &\longleftarrow A_{12}/2 = 5 \\ B_{13} &\longleftarrow B_{12} = 19 \\ C_{13} &\longleftarrow C_{12} = 4 \end{aligned}$$

Step 14: A_{13} und B_{13} sind beide ungerade, daher:

$$\begin{aligned} A_{14} &\longleftarrow\mid A_{13} - B_{13} \mod = 14 \\ B_{14} &\longleftarrow \min\{A_{13}, B_{13}\} = 5 \\ C_{14} &\longleftarrow C_{13} = 4 \end{aligned}$$

Step 15: A_{14} ist gerade und B_{14} ist ungerade, daher:

$$\begin{aligned} A_{15} &\longleftarrow A_{14}/2 = 7 \\ B_{15} &\longleftarrow B_{14} = 5 \\ C_{15} &\longleftarrow C_{14} = 4 \end{aligned}$$

Step 16: Die beiden Zahlen A_{15} und B_{15} sind ungerade, wir setzen:

$$\begin{aligned} A_{16} &\longleftarrow\mid A_{15} - B_{15} \mid = 2 \\ B_{16} &\longleftarrow \min\{A_{15}, B_{15}\} = 5 \\ C_{16} &\longleftarrow C_{15} = 4 \end{aligned}$$

Step 17: A_{16} ist gerade und B_{16} ist ungerade, daher:

$$\begin{aligned} A_{17} &\longleftarrow A_{16}/2 = 1 \\ B_{17} &\longleftarrow B_{16} = 5 \\ C_{17} &\longleftarrow C_{16} = 4 \end{aligned}$$

Step 18: Da die beiden Zahlen A_{17} und B_{17} ungerade sind, setzen wir:

$$\begin{aligned} A_{18} &\longleftarrow \mid A_{17} - B_{17} \mid = 4 \\ B_{18} &\longleftarrow \min\{A_{17}, B_{17}\} = 1 \\ C_{18} &\longleftarrow C_{17} = 4 \end{aligned}$$

Step 19: Da A_{19} gerade und B_{19} ungerade ist, folgt

$$\begin{aligned} A_{19} &\longleftarrow A_{18}/2 = 2 \\ B_{19} &\longleftarrow B_{18} = 1 \\ C_{19} &\longleftarrow C_{18} = 4 \end{aligned}$$

Step 20: Da A_{19} gerade und B_{19} ungerade ist, folgt:

$$\begin{aligned} A_{20} &\longleftarrow A_{19}/2 = 1 \\ B_{20} &\longleftarrow B_{19} = 1 \\ C_{20} &\longleftarrow C_{19} = 4 \end{aligned}$$

Step 21: Da $A_{20} = B_{20}$ terminiert der Algorithmus und $\texttt{ggT}(2152, 764) = C_{20} = 4$.

Bei der Implementierung in einer Programmiersprache erfordert der STEIN Algorithmus keine Divisionsinstruktionen. Die ausgeführten Operationen sind

- Subtraktionen
- Paritätsprüfung
- Halbierung von geraden Zahlen

Gerade die letzte Operation läßt sich sehr effektiv als Right-Shift Operation umsetzen.

18.3.1 Übungen

Übung 18.2:

Berechnen Sie $d = \mathrm{ggT}(360, 294)$ auf drei Arten:

1. durch Zerlegung der beiden Zahlen in ihre Primfaktoren und daraus die Primfaktor Zerlegung von d.
2. mit Hilfe des EUKLIDischen Algorithmus.
3. mit Hilfe der STEIN Algorithmus.

Übung 18.3:

Berechnen Sie für jedes der folgenden Paare von Zahlen den größten gemeinsamen Teiler mit Hilfe des EUKLIDischen Algorithmus. Drücken Sie den ggT dann aus als Linearkombination dieser beiden Zahlen.

1. 26, 19
2. 187, 34
3. 841, 160
4. 2613, 2171

Übung 18.4:

Berechnen Sie mit Hilfe des erweiterten EUKLIDischen Algorithmus das multiplikative Inverse

$$19^{-1} \bmod 251.$$

Übung 18.5:

Berechnen Sie das multiplikative Inverse

$$33^{-1} \bmod 742.$$

Kapitel 19

Primzahlen

In diesem Kapitel sehen wir uns eine Reihe von Eigenschaften von Primzahlen an, die für kryptographische Systeme interessant sind.

19.1 Faktorisierung

Die **Faktorisierung** einer Zahl bedeutet, ihre Zerlegung in Potenzen von Primzahlen zu finden. Hier sind einige Beispiele:

$$\begin{aligned} 28 &= 2 \cdot 2 \cdot 7 \\ 60 &= 2 \cdot 2 \cdot 3 \cdot 5 \\ 252601 &= 41 \cdot 61 \cdot 101 \end{aligned}$$

Das Faktorisierungsproblem ist ein sehr altes Problem, mit dem sich bereits die Griechen beschäftigt haben. Wichtig zu verstehen ist, dass es prinzipiell sehr einfach ist, eine Zahl zu faktorisieren. Das Problem liegt darin, dass alle bekannten Verfahren *sehr* zeitaufwändig sind. Diese Aussage ist heute nach wie vor gültig, obwohl es mittlerweile eine Reihe von Verbesserungen gibt.

Die folgende (unvollständige) Liste gibt einige Faktorisierungsverfahren:

- **Das Zahlenfeldsieb**
  Das allgemeine Zahlenfeldsieb ist der schnelleste bekannte Algorithmus zur Faktorisierung von Zahlen mit mehr als 110 Stellen.
- **Das quadratische Zahlenfeldsieb**
  Das quadratischen Zahlensieb ist der schnellste bekannte Algorithmus zur Faktorisierung von Zahlen mit weniger als 110 Stellen (siehe [52, Chap.6.1.]).
- **Elliptische Kurven Methoden**
  Siehe Kapitel [11.4].
- **Pollards Monte Carlo Methode**
  Diese Methode werden wir in Kapitel [19.1.3] untersuchen.
- **Teilermethode**
  Die Teilermethode ist der älteste Faktorisierungsalgorithmus und besteht

im wesentlichen daraus, zu testen, ob die Primzahlen kleiner der Quadratwurzel der gegebenen Zahl diese Zahl teilen oder nicht.

Um ein Gefühl für die Problematik zu erhalten, wie aufwändig die Faktorisierung großer Zahlen ist, hier ein kurzer Rückblick.

FERMAT Zahlen

Der französische Mathematiker PIERRE DE FERMAT (1601 – 1665) behauptete, dass alle Zahlen der Form

$$\mathcal{F}_i = 2^{2^i} + 1$$

Primzahlen sind.

In der Tat, $\mathcal{F}_0 = 3, \mathcal{F}_1 = 5, \mathcal{F}_2 = 17, \mathcal{F}_3 = 257$ und $\mathcal{F}_4 = 65537$ sind alles Primzahlen. Im Jahre 1732 entdeckte LEONHARD EULER, dass $\mathcal{F}_5 = 4.294.967.297$ keine Primzahl ist, sondern das Produkt von 641 und 6.700.417. Mittlerweile weiß man auch, dass die FERMAT Zahlen $\mathcal{F}_6, \mathcal{F}_7, \mathcal{F}_8$ und $\mathcal{F}_9$ ebenfalls zusammengesetzt sind. Die Faktorisierung von $\mathcal{F}_6$ wurde 1880, die Faktorisierung von $\mathcal{F}_7$ im Jahre 1970 gefunden. Zehn Jahre später gelang es BRENT und POLLARD die FERMAT Zahl $\mathcal{F}_8$ zu faktorisiern, im Jahre 1990 wurde $\mathcal{F}_9$ faktorisiert.

Die Lehre aus dieser Entwicklung ist die folgende:

- es dauerte bis in das Jahr 1970 um die 39–stellige FERMAT Zahl $\mathcal{F}_7$ zu faktorisieren;
- aufgrund der enormen Verbesserung der Rechnerleistung und neuen Faktorisierungsalgorithmen dauerte es nur 20 Jahre, bis die Faktorisierung der 155–stelligen FERMAT Zahl $\mathcal{F}_9$ gelang.

Das Argument, dass die Faktorisierung ein 'hartes' Problem ist, geht folgendermaßen:

① Eine zu faktorisierende Zahl N erfordert

$$n = \lfloor \log_2 N \rfloor + 1$$

Bits zur Codierung in Rechnern.

② Offensichtlich hat eine zusammengesetzte Zahl N einen Faktor im Zahlenintervall $[1, \sqrt{N}]$.

③ Testet man nun jede Zahl in diesem Intervall um einen potentiellen Teiler der Zahl N zu finden, erfordert dies

$$\sqrt{N} = 2^{n/2} \in O(2^n)$$

Schritte. Mit anderen Worten, dies ist also ein exponentielles Wachstumsverhalten der Laufzeit eines Faktorisierungsalgorithmus mit der Länge des Inputs, *i.e.* der zu faktorisierenden Zahl N.

Die besten gegenwärtig bekannten Faktorisierungsalgorithmen (Quadratisches Sieb) haben ein Laufzeitverhalten von

$$O(\left(e^{\sqrt{2 \log n \log \log n}}\right).$$

19.1.1 Die RSA Challenge

Als im Jahre 1977 der RSA Algorithmus von RON RIVEST, ADI SHAMIR und LEO ADLEMAN publiziert wurde, stellten die Entwickler die folgende Aufgabe an die Mathematiker–Gemeinde: (siehe [90, 215])

> Der RSA-Modul sei:
>
> $$n = 114381625757888867669235779961466120102182967212423625625618429357069352457338978305971235639587050589890751475992900268795435 41$$
>
> und sei $e = 9001$ der Verschlüsselungsexponent (*i.e.* $K_{\text{pub}} = (e, n)$).
> Der Chiffretext sei:
>
> $$C = 968696137546220614771409222543558829057599911245743198746951209308162982251457083569314766228839896280133919905518299451578151 54$$
>
> Ermittle den Klartext.

Der einzige Weg, den Klartext zu erhalten, führt über die Faktorisierung von n. RIVEST, SHAMIR und ADLEMAN bestimmten, dass mit den damals zur Verfügung stehenden Faktorisierungsalgorithmen etwa 10^{16} Jahre ins Land gehen, bis jemand dieses Ziel erreicht hat. Aus diesem Grund wagten die Autoren, 100$ Preisgeld auszusetzen, dem es bis zum 1. April 1982 gelingt, den Klartext zu erhalten. Die 10^{16} Jahre waren im Jahre 1994 um, denn es gelang ATKINS, GRAFF, LENSTRA und LEYLAND[1] die Zahl 128-stellige Zahl n zu faktorisieren (siehe auch [162]).

Seit 1991 hat die Firma RSA Security einen Wettbewerb ausgeschrieben, die sogenannte **RSA Challenge**. Dieser Wettbewerb endete Im Mai 2007. Zweck dieses Wettbewerbs war es, Erfahrungen über die Schwierigkeit der Faktorisierung großer Zahlen — ähnlich der Zahlen in den RSA Schlüsseln — zu sammeln. Es waren acht Challenges ausgeschrieben mit Zahlen von der Länge 576 bis 2.048 Bits. Jede dieser Zahlen ist das Produkt zweier großen Primzahlen. Ziel ist es jeweils, die beiden Primfaktoren anzugeben.

Die größte Zahl, die im Rahmen des **RSA–155** Challenges faktorisiert wurde, ist eine 512 Bit Zahl[2].

Die folgenden Herausforderungen wurden an die Kryptogemeinde gestellt:

- Name: **RSA-576**
  Preisgeld: $10 000

[1] Siehe: D. ATKINS, M. GRAFF, A. K. LENSTRA und P.C. LEYLAND, *The magic words are squeamish ossifrage*, pp. 263 – 277 in *Advances in Cryptology – ASIACRYPT '94*, J. Pieprzyk and R. Safavi-Naini, eds. Lecture Notes in Comp. Sci. 917, Springer 1995. Siehe auch [226].

[2] Details siehe die URL:

http://www.rsasecurity.com/rsalabs/node.asp?id=2098#

Anzahl Stellen (Dezimal): 174
Digit Sum: 785

188198812920607963838697239461650439807163563379417382700763356422988859715234665485319060606504743045317388011303396716199692321205734031879550656996221305168759307650257059

- Name: RSA-640
  Preisgeld : $20 000
  Anzahl Stellen : 193
  Digit Sum: 806

  3107418240490043721350750035888567930037346022842727545720161948823206440518081504556346829671723286782437916272838033415471073108501919548529007337724822783525742386454014691736602477652346609

- Name: **RSA-704**
  Preisgeld: $30 000
  Anzahl Dezimalstellen: 212
  Digit Sum: 1009

  740375634795617128280467960974295731425931888892312890849362326389727650340282662768919964196251178439958943305021275853701189680982867331732731089309005525051168770632990723963807867100860969625379346505637963 59

- Name: **RSA-768**
  Preisgeld: $50 000
  Dezimalstellen: 232
  Digit Sum: 1018

  123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3

- Name: **RSA-896**
  Preisgeld: $75 000
  Anzahl Dezimalstellen: 270
  Digit Sum: 1222

  412023436986659543855531365332575948179811699844327982845455626433876445565248426198098870423161841879261420247188869492560931776375033421130982397485150944909106910269861031862704114880866970564902903653658867433731720813104105190864254793282601391257624033946373269391

- Name: **RSA-1024**
  Preisgeld: $100 000
  Anzahl Dezimalstellen: 309
  Digit Sum: 1369

  135066410865995223349603216278805969938881475605667027524485143
  851526510604859533833940287150571909441798207282164471551373680
  419703964191743046496589274256239341020864383202110372958725762
  358509643110564073501508187510676594629205563685529475213500852
  879416377328533906109750544334999811150056977236890927563

- Name: **RSA-1536**
  Preisgeld: $150 000
  Anzahl Dezimalstellen: 463
  Digit Sum: 2153

  184769970321174147430683562020016440301854933866341017147178
  577491065169671116124985933768430543574458561606154457179405
  222971773252466096064694607124962372044202226975675668737842
  756238950876467844093328515749657884341508847552829818672645
  133986336493190808467199043187438128336350279547028265329780
  293491615581188104984490831954500984839377522725705257859194
  499387007369575568843693381277961308923039256969525326162082
  367649031603655137144791393234716956698806 9

- Name: **RSA-2048**
  Preisgeld: $200 000
  Anzahl Dezimalstellen: 617
  Digit Sum: 2738

  251959084756578934940271832400483985714292821262040320277771
  378360436620207075955562640185258807844069182906412495150821
  892985591491761845028084891200728449926873928072877767359714
  183472702618963750149718246911650776133798590957000973304597
  488084284017974291006424586918171951187461215151726546322822
  168699875491824224336372590851418654620435767984233871847744
  479207399342365848238242811981638150106748104516603773060562
  016196762561338441436038339044149526344321901146575444541784
  240209246165157233507787077498171257724679629263863563732899
  121548314381678998850404453640235273819513786365643912120103
  97122822120720357

19.1.2 Die Teilermethode

Um zu sehen, wie die Faktorisierung einer Zahl funktioniert, sehen wir uns hier im Detail die Teilermethode zur Faktorisierung an ([126]). Die grundlegende

Idee dieses Verfahrens besteht darin, eine Zahl durch Division mit Testteilern in das Produkt ihrer Primfaktoren zu zerlegen. Falls $N > 1$, dann wird die Zahl N durch aufeinanderfolgende Primzahlen $p = 2, 3, 5, 7, \ldots$ so lange geteilt, bis das kleinste p gefunden ist, so dass

$$N \bmod p \equiv 0 \bmod p.$$

Damit ist diese Primzahl der kleinste Primfaktor von N. Dann wird dieses Prozeder wiederholt mit $N \leftarrow N/p$ und dieser neue Wert wird durch p und höhere Primzahlen geteilt. Finden wir in allen Schritte, dass

$$N \bmod p \not\equiv 0 \bmod p,$$

und $\lfloor N/p \rfloor \leq p$, dann muß N selbst Primzahl sein.

Formal lautet der Algorithmus wie folgt:

Algorithmus Teilermethode

Sei N eine positive ganze Zahl. Der folgende Algorihmus findet die Primfaktoren $p_1 \leq p_2 \leq p_3 \cdots \leq p_t$ von N. Das Verfahren benutzt eine Menge von Teilern mit

$$2 = d_0 < d_1 < d_2 < \cdots,$$

die alle Primzahlen $\leq \sqrt{N}$ enthält. Diese Menge kann Zahlen enthalten, die keine Primzahlen sind.

FAK0 *Input*
Der Algorithmus benutzt die folgenden Variablen:

- n Variable für die zu faktorisierende Zahl
- q Variable für die ganzzahlige Division
- r Variable für den ganzzahligen Rest der Division
- t Zählvariable für die Liste der Primfaktoren
- k Zählvariable für die Liste der Teiler

FAK1 *Initialisierung*
Setze

$$t \leftarrow 0$$
$$k \leftarrow 0$$
$$n \leftarrow N$$

FAK2 $n = 1$ *?*
Falls $n = 1$, dann terminiert der Algorithmus.

FAK3 *Division*
Setze

$$q = \lfloor n/d_k \rfloor$$
$$r = n \mod d_k$$

q und r bezeichnen den ganzzahligen Quotienten und den Rest, wenn n durch den Faktor d_k dividiert wird.

FAK4 *Rest = 0?*
Falls $r \neq 0$ gehe zu **FAK6**.

FAK5 *Faktor gefunden*
Erhöhe t um 1 und setze $d_k \rightarrow p_t$ und $q \rightarrow n$. Weiter mit Schritt **FAK2**.

FAK6 *Kleiner Quotient?*
Falls $q > d_k$, dann erhöhe den Zähler k um 1 gehe zum Schritt **FAK3**. Sonst weiter mit **FAK7**.

FAK7 *n ist Primzahl*
Erhöhe t um 1 und setze $p_t \rightarrow n$. STOP, der Algorithmus terminiert.

Die Abbildung [19.1] zeigt den Ablauf dieses Algorithmus in einem Flußdiagramm.

Beispiel [19.50]

Um zu verstehen, wie dieser Algorithmus arbeitet, ist es zweckmäßig, ein Beispiel im Detail auszuarbeiten. Daher benutzen wir diesen Algorithmus, um die Zahl $N = 25852$ zu faktorisieren.

Step 1 $\rightarrow$ **FA0** Input
$N = 25852$ und eine Liste von Teilern:

$d_0 = 2$	$d_1 = 3$
$d_2 = 5$	$d_3 = 7$
$d_4 = 11$	$d_5 = 13$
$d_6 = 17$	$d_7 = 19$
$d_8 = 23$	$d_9 = 25$
$d_{10} = 29$	$d_{11} = 31$
$d_{12} = 35$	$d_{13} = 37$

Anmerkung: Diese Liste sollte alle Primzahlen p enthalten mit $p \leq \sqrt{N}$.

Step 2 $\rightarrow$ **FAK1** Initialisierung
Wir setzen den Index t – diese Variable listet die gefundenen Primfaktoren auf – auf den Wert 0. Weiterhin beginnen wir mit dem ersten Element der Teilerliste, i.e. wir setzen $k = 0$ und n auf den Wert $N = 25852$.

Step 3 $\rightarrow$ **FAK2** $n = 1$? $\rightarrow$ Nein, daher weiter mit **FAK3**.

Step 4 $\rightarrow$ **FAK3** Division
Berechne

$$q = \lfloor n/d_0 \rfloor = \lfloor 25852/2 \rfloor = 12926$$

und

$$r = n \mod d_0 = 25852 \mod 2 = 0$$

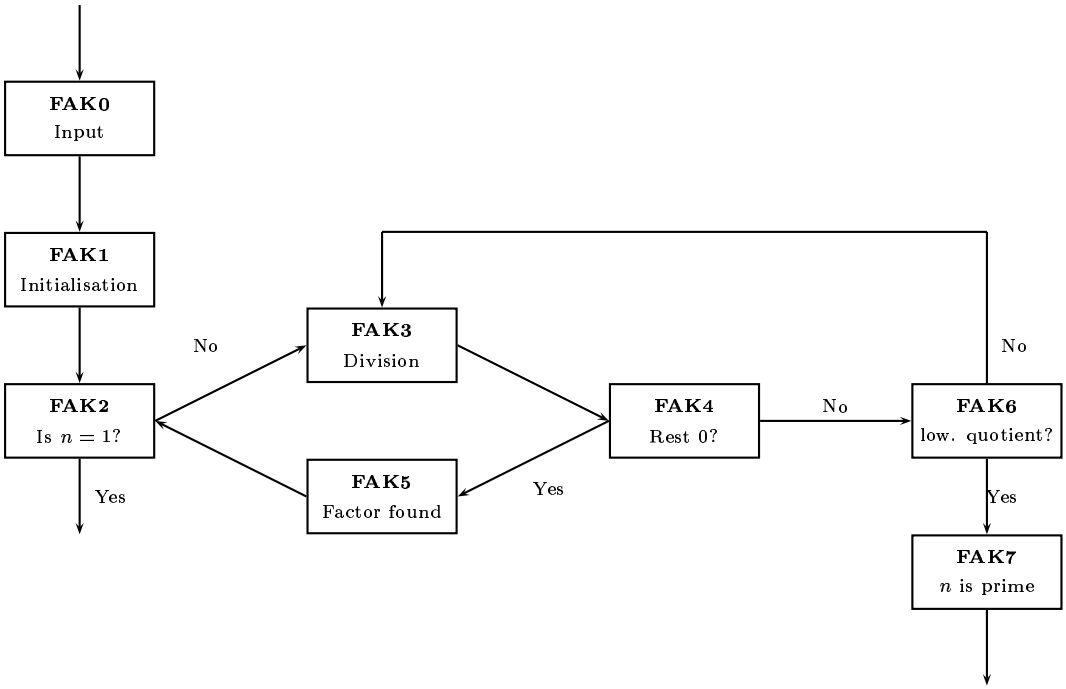


Abbildung 19.1: Flow diagram of the trial division algorithm.

Step 5 $\rightarrow$ **FAK4** Divisionsrest
da $r = 0$ weiter mit **FAK5**.

Step 6 $\rightarrow$ **FAK5** Faktor gefunden
Setze $t = 1$ und

$$p_1 = d_0 = 2.$$

Redefinition von n; setze $n = 12926$. Weiter mit Schritt **FAK2**.

Step 7 $\rightarrow$ **FAK2** $n = 1$? $\rightarrow$ Nein, daher weiter mit **FAK3**.

Step 8 $\rightarrow$ **FAK3** Division
Berechne

$$q = \lfloor n/d_0 \rfloor = \lfloor 12926/2 \rfloor = 6463$$

und

$$r = n \mod d_0 = 12926 \mod 2 = 0$$

Step 9 $\rightarrow$ **FAK4** Divisionsrest
Da $r = 0$ weiter mit Schritt **FAK5**.

Step 10 → **FAK5** Faktor gefunden
Setze $t = 2$ und

$$p_2 = d_0 = 2.$$

Redefinition von n; setze $n = 6463$. Weiter mit dem Schritt **FAK2**.

Step 11 → **FAK2** $n = 1$? → nein, weiter mit **FAK3**.

Step 12 → **FAK3** Division
Berechne

$$q = \lfloor n/d_0 \rfloor = \lfloor 6463/2 \rfloor = 3231$$

und

$$r = n \mod d_0 = 6463 \mod 2 = 1$$

Step 13 → **FAK4** Betrachte Divisionsrest:
Da $r \neq 0$ weiter mit Schritt **FAK6**.

Step 14 → **FAK6** Untersuche den Quotienten
Da $q = 3231 > d_0 = 2$ erhöhe den Index der Teilerliste um 1: $k = 1$; weiter mit Schritt **FAK3**.

Step 15 → **FAK3** Division
Berechne

$$q = \lfloor n/d_1 \rfloor = \lfloor 6463/3 \rfloor = 2154$$

und

$$r = n \mod d_1 = 6463 \mod 3 = 1$$

Step 16 → **FAK4** Divisionsrest:
Da $r \neq 0$ weiter mit Schritt **FAK6**.

Step 17 → **FAK6** Untersuche den Quotienten:
Da $q = 2154 > d_1 = 3$ erhöhe den Index der Teilerliste um 1: $k = 2$; weiter mit Schritt **FAK3**

Step 18 → **FAK3** Division
Berechne

$$q = \lfloor n/d_2 \rfloor = \lfloor 6463/5 \rfloor = 1292$$

und

$$r = n \mod d_2 = 6463 \mod 5 = 3$$

Step 19 → **FAK4** Divisionsrest:
Da $r \neq 0$, ist $d_2 = 5$ kein Primfaktor, weiter mit **FAK6**.

Step 20 → **FAK6** Quotient
Da $q = 1292 > d_2 = 5$, gehe zum nächsten Teiler; $k = 3$; weiter mit Schritt **FAK3**

Step 21 → **FAK3** Division
Berechne

$$q = \lfloor n/d_3 \rfloor = \lfloor 6463/7 \rfloor = 923$$

und

$$r = n \mod d_3 = 6463 \mod 7 = 2$$

Step 22 → **FAK4** Divisionsrest:
Da $r \neq 0$, ist $d_3 = 7$ ebenfalls kein Primfaktor, daher weiter mit Schritt **FAK6**.

Step 23 → **FAK6** Quotient untersuchen:
Da $q = 923 > d_3 = 7$ erhöhe k um 1: $k = 4$; weiter mit Schritt **FAK3**

Step 24 → **FAK3** Division
Berechne

$$q = \lfloor n/d_4 \rfloor = \lfloor 6463/11 \rfloor = 587$$

und

$$r = n \mod d_4 = 6463 \mod 11 = 5$$

Step 25 → **FAK4** Divisionsrest:
Da $r \neq 0$ weiter mit **FAK6**.

Step 26 → **FAK6** Quotient
Da $q = 587 > d_4 = 11$ erhöhe den Index der Teilerliste um 1: $k = 5$; weiter mit **FAK3**

Step 27 → **FAK3** Division
Berechne

$$q = \lfloor n/d_5 \rfloor = \lfloor 6463/13 \rfloor = 497$$

und

$$r = n \mod d_5 = 6463 \mod 13 = 2$$

Step 28 → **FAK4** Divisionsrest:
Da $r \neq 0$ weiter mit **FAK6**.

Step 29 → **FAK6** Quotient
Da $q = 497 > d_5 = 13$ setze: $k = 6$; weiter mit **FAK3**.

Step 30 → **FAK3** Division
Berechne

$$q = \lfloor n/d_6 \rfloor = \lfloor 6463/17 \rfloor = 380$$

und

$$r = n \mod d_6 = 6463 \mod 17 = 3$$

Step 31 → **FAK4** Divisionsrest:
Da $r \neq 0$, ist $d_6 = 17$ ebenfalls kein Primfaktor, daher weiter mit **FAK6**.

Step 32 → **FAK6** Quotient
Since $q = 380 > d_6 = 17$ wird der nächste Teiler untersucht: $k = 7$; weiter mit **FAK3**.

Step 33 → **FAK3** Division
Berechne

$$q = \lfloor n/d_7 \rfloor = \lfloor 6463/19 \rfloor = 340$$

und

$$r = n \mod d_7 = 6463 \mod 19 = 3$$

Step 34 → **FAK4** Divisionsrest:
Da $r \neq 0$ weiter mit **FAK6**.

Step 35 → **FAK6** Untersuche den Quotienten
Da $q = 340 > d_7 = 19$ teste den nächsten Teiler aus der Liste: $k = 8$; weiter mit **FAK3**.

Step 36 → **FAK3** Division
Berechne

$$q = \lfloor n/d_8 \rfloor = \lfloor 6463/23 \rfloor = 281$$

und

$$r = n \mod d_8 = 6463 \mod 23 = 0$$

Step 37 → **FAK4** Divisionsrest
Da $r = 0$ ist ein Primfaktor gefunden, weiter mit Schritt **FAK5**.

Step 38 → **FAK5** Faktor gefunden
Setze $t = 3$ und

$$p_3 = d_8 = 23.$$

Redefinition von n; die folgenden Schritte sind mit $n = 281$ durchzuführen; weiter mit **FAK2**.

Step 39 → **FAK2** $n = 1$? → nein, weiter mit **FAK3**.

Step 40 $\rightarrow$ **FAK3** Division
Berechne

$$q = \lfloor n/d_8 \rfloor = \lfloor 281/23 \rfloor = 12$$

und

$$r = n \mod d_8 = 281 \mod 23 = 5$$

Step 41 $\rightarrow$ **FAK4** Divisionsrest untersuchen
Da $r \neq 0$ ist $d_8 = 23$ nicht nochmals Faktor, daher weiter mit **FAK6**.

Step 42 $\rightarrow$ **FAK6** Betrachte den Quotienten
Da $q = 12 < d_8 = 23$ weiter mit Schritt **FAK7**

Step 43 $\rightarrow$ **FAK7** n ist selbst Primfaktor
Erhöhe t um 1 und setze $p_4 = 281$. STOP

Daher lautet die Faktorisierung von $N = 25852$:

$$25852 = p_1 \cdot p_2 \cdot p_3 \cdot p_4 = 2 \cdot 2 \cdot 23 \cdot 281$$

Um die Primfaktoren der Zahl $N = 25852$ zu finden, benötigt man mit diesem Verfahren 12 Divisionsoperationen.

19.1.3 Pollards Rho Algorithmus

Eine alternative Möglichkeit, Zahlen zu faktorisieren ist die von JOHN M. POLLARD im Jahre 1975 entwickelte ρ Methode.[3] [171]

Sei $n > 3$ eine zusammengesetzte Zahl mit Primfaktoren $p_1, p_2, \ldots, p_r$. Weiter seien zwei Zahlen x, y gegeben mit

$$x \not\equiv y \bmod n.$$

Es kann aber vorkommen, dass

$$x \equiv y \bmod p_i, \quad \text{für ein } i \in \{1, 2, \ldots, r\}.$$

Um festzustellen, ob x und y diese Eigenschaft haben, müssen wir prüfen, ob p_i Teiler von $x - y$ ist. Dies ist aber das Problem, denn wir wissen zwar, dass n zusammengesetzt ist[4], kennen jedoch nicht die Primfaktoren.

In diesem Fall bietet es sich an, den größten gemeinsamen Teiler

$$d = \mathrm{ggT}(x - y, n)$$

zu berechnen. Die Argumentation ist nun wie folgt: Genau dann ist $d > 1$, wenn die Differenz $x - y$ durch einen Primfaktor von n teilbar ist und d ist dann solch ein Primfaktor oder das Produkt von solchen.

Hier setzt nun die erste Idee von POLLARD an. Man wähle ein zufälliges k-Tupel von Zahlen $0 \leq x_\nu < n, \nu = 1, 2 \ldots, k$. Dabei ist k so groß zu wählen, dass die Wahrscheinlichkeit hoch ist, dass

$$x_\nu \equiv x_\mu \bmod p_i$$

für wenigstens ein Paar ν, μ. Dies ist eine unmittelbare Anwendung des Geburtstagsparadoxon, das wir in Abschnitt [10.3.3] betrachtet haben. Zur Wiederholung:

Satz

Sei M eine Menge mit m paarweise voneinander verschiedenen Elementen. Aus M werden $k \leq m$ Stichproben mit Zurücklegen genommen. Dann ist die Wahrscheinlichkeit $P_{m,k}$ dafür, dass alle Stichproben paarweise verschieden sind, gleich:

$$P_{m,k} = \prod_{i=1}^{k-1} \left(1 - \frac{i}{m}\right). \tag{19.1}$$

Der Beweis dieses Satzes wird mittels vollständiger Induktion ausgeführt (siehe [85, p. 105]).

[3] Siehe die Darstellungen in [85, pp. 105 – 112] oder [52], [51].

[4] Dies ist bekannt durch Anwenden eines Primzahltests wie der SOLOVAY-STRASSEN Test auf die Zahl n.

Beispielsweise erhält man für $m = 365, k = 30$ den Wert:

$$P_{365,30} = 0.2936\ldots .$$

Dies ist genau das *Geburtstagsparadoxon*: Betrachtet man eine Menge von 30 Personen, so ist die Wahrscheinlichkeit dafür, dass zwei davon am selben Tag im Jahr Geburtstag haben, mehr als 70 Prozent. Bei 50 Personen liegt diese Wahrscheinlichkeit bei 97 %.

Gemäß Gl. (10.1) kann Gl. (19.1) durch

$$P_{m,k} \approx e^{-\frac{k(k-1)}{2m}}$$

approximiert werden.

Alternativer Beweis:

Wir logarithmieren Gl. (19.1) und erhalten:

$$\log P_{m,k} = \sum_{i=1}^{k} \log(1 - \frac{i}{m}).$$

Für $m >> k$ approximiert man

$$\log(1-x) \approx -x$$

und es folgt:

$$-\log P_{m,k} \approx \sum_{i=1}^{k-1} \frac{i}{m} = \frac{k(k-1)}{2m}$$

oder

$$P_{m,k} \approx e^{-\frac{k(k-1)}{2m}}.$$

Der kritische Wert $P_{m,k} = \frac{1}{2}$ wird erreicht bei $k \approx 1.2\sqrt{m}$.

Wenden wir dieses Ergebnis auf das ursprüngliche Problem an. Ist p ein Primteiler von n und ist ein zufälliges k-Tupel von Zahlen $0 \leq x_\nu < n$ vorgegeben, wobei $k = c \cdot \sqrt{p}$, c eine kleine Konstante $c > 1$. Gemäß dem obigen Satz und den Folgerungen daraus, können wir erwarten, dass zwei dieser Zahlen kongruent modulo p sind. Allerdings ist nicht bekannt, welche der Zahlen aus dem Tupel das sind. Berechnet man deshalb für alle Paare ν, μ den Wert

$$\mathrm{ggT}(x_\nu - x_\mu, n),$$

dann kommt man wieder auf p Operationen und man hat nichts gewonnen.

Um dieses Problem in den Griff zu bekommen, wendet POLLARD die folgende Methode an. Die zufälligen Zahlen x_ν werden als Pseudo-Zufallsfolge rekursiv aus einem Startwert x_0 durch die Vorschrift

$$x_{i+1} = f(x_i)$$

berechnet. Dabei ist f eine Abbildung

$$f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n.$$

Dann gilt folgender Satz:

Satz

Sei

$$f : \mathbb{M} \longrightarrow \mathbb{M}$$

eine Abbildung einer endlichen Menge $\mathbb{M}$ auf sich. Für einen beliebigen Anfangswert $x_0 \in \mathbb{M}$ wird eine Folge rekursiv definiert über

$$x_{i+1} = f(x_i)$$

Dann gibt es ganze Zahlen $n_0 \geq 0, k > 0$ mit

$$x_{i+k} = x_i \text{ für alle } i \geq n_0$$

Seien n_0 und k minimal gewählt. Dann heißen n_0 die *Vorperiode* und k die *Periode* der Folge (x_i). Vergleicht man x_i und x_{2i} der Reihe nach für $i = 1, 2, 3, \ldots$, dann existiert ein minimales $i_0 > 0$ mit

$$x_{i_0} = x_{2i_0}$$

Diese Zahl i_0 ist ein ganzzahliges Vielfaches der Periode k. Ist $n_0 \leq k$ so ist $i_0 = k$. In jedem Fall ist i_0 das kleinste Vielfache der Periode k mit $i_0 \geq n_0$.

Man kann die Menge M mit der derart definierten Abbildung f als ein diskretes dynamisches System interpretieren. Die mit dem Anfangswert x_0 definierte Folge

$$x_0, x_1, x_2, \ldots, x_i, \ldots$$

heißt die *Bahn* des Punktes x_0. Den Index i kann man als diskreten Zeitparameter auffassen. In dieser Interpretation besagt obiger Satz, dass die Bahn nach dem Durchlaufen einer Vorperiode in einen Zyklus einläuft. Die Gestalt der Bahn hat die Form des griechischen Buchstabens ρ, daher der Name POLLARD's Rho Methode (siehe die Abbildung [19.2]).

Beweis:

Da die Menge M endlich ist, können nicht alle x_i verschieden sein. Es gibt also ein $n_0 \geq 0$ und ein $k > 0$, so dass

$$x_{n_0+k} = x_{n_0}.$$

Hieraus folgt aber durch wiederholtes Anwenden der Abbildung f, dass

$$x_{i+k} = x_i \qquad \forall\, i \geq n_0.$$

Damit ist der erste Teil des Satzes bewiesen.
Wählt man $k > 0$ minimal, dann folgt für jedes $k' > 0$ mit

$$x_{j+k'} = x_j$$

dass k' ein ganzzahliges Vielfaches von k ist. Anderfalls hätte man

$$k' = q \cdot k + k''$$

mit ganzen Zahlen q und k'' mit $0 < k'' < k'$. Es würde dann folgen

$$x_{l+k''} = x_l$$

für genügend großes l, was der Minimalität von k widerspricht. Daher folgt aus $x_{i_0} = x_{2i_0}$, dass $i_0 = 2i_0 - i_0$ ein ganzzahliges Vielfaches der Periode ist. Sei $i_0 = q \cdot k$ und die Zahl ν so groß gewählt, dass $i_0 + \nu qk$ größer oder gleich der Vorperiode ist. Dann gilt

$$x_{i_0+\nu qk+k} = x_{i_0+\nu qk}.$$

Die linke Seite ist gleich x_{i_0+k}, die rechte Seite ist gleich x_{i_0}, also folgt $x_{i_0+k} = x_{i_0}$. Für die minimale Vorperiode n_0 gilt daher $i_0 \geq n_0$. Umgekehrt folgt natürlich für $i_0 = qk \geq n_0$ mit ganzzahligem q, dass $x_{i_0} = x_{2i_0}$.

Algorithmus Faktorisierung mit Pollard's Rho Methode

Gegeben ist eine zusammengesetzte Zahl n.
Der folgende Algorithmus versucht, einen nichttrivialen Faktor von n zu finden.

P0 *Startwerte auswählen:*
Wahle einen Wert $s \in \mathbb{Z}_n$.
Setze $x_1 = y \longleftarrow s$.
Wähle

$$F(x) = (x^2 - 1) \bmod n$$

P1 *Faktorsuche:*
Berechne

$$x_{i+1} \longleftarrow F(x_i).$$

Setze

$$y \longleftarrow F(F(y))$$

Berechne

$$d = \mathrm{ggT}(y - x_{i+1}, n).$$

Falls $d = 1$ wiederhole P1.

P2 *Ungeeigneter Startwert:*
Falls $d = n$, gehe zu P0.

P3 *Faktor gefunden:*
STOP, return d.

Beispiel:

Um sich mit dem Faktorisierungsalgorithmus von POLLARD vertraut zu machen, betrachten wir folgendes Beispiel:

$$n = 1387$$

mit Startwert $s = x_1 = y = 2$ und der Funktion

$$F(x) = (x^2 - 1) \bmod n.$$

Schritt 1: Berechne

$$x_2 \longleftarrow F(x_1) = (x_1^2 - 1) \bmod n = 3 \bmod 1387.$$

und

$$\begin{aligned} y \longleftarrow F(F(y)) &= F[(y^2-1) \bmod 1387] \\ &= F(3) \\ &= (3^2-1) \bmod 1387 \\ &= 8 \end{aligned}$$

Berechne

$$d = \mathrm{ggT}(8-3, 1387) = 1.$$

Schritt 2: Berechne

$$x_3 \longleftarrow F(x_2) = (x_2^2 - 1) \bmod n = 8 \bmod 1387.$$

und

$$\begin{aligned} y \longleftarrow F(F(y)) &= F[(y^2-1) \bmod 1387] \\ &= F(63) \\ &= (63^2-1) \bmod 1387 \\ &= 1194 \end{aligned}$$

Berechne

$$d = \mathrm{ggT}(1194-8, 1387) = 1.$$

Schritt 3: Berechne

$$x_4 \longleftarrow F(x_3) = (x_3^2 - 1) \bmod n = 63 \bmod 1387.$$

und

$$\begin{aligned} y \longleftarrow F(F(y)) &= F[(y^2-1) \bmod 1387] \\ &= F[1194^2 - 1 \bmod 1387] \\ &= F(1186) \\ &= (1186^2-1) \bmod 1387 \\ &= 177. \end{aligned}$$

Berechne

$$d = \mathrm{ggT}(177-63, 1387) = 19.$$

Hier bricht der Algorithmus ab, denn einer der beiden Faktoren ist gefunden:

$$1387 = 19 \times 73.$$

Die folgenden Glieder der Iteration sind:

$$\begin{aligned}
x_5 &= F(x_4) = (63^2 - 1) \bmod 1387 = 1194\\
x_6 &= F(x_5) = (1194^2 - 1) \bmod 1387 = 1186\\
x_7 &= F(x_6) = (1186^2 - 1) \bmod 1387 = 177\\
x_8 &= F(x_7) = (177^2 - 1) \bmod 1387 = 814\\
x_9 &= F(x_8) = (814^2 - 1) \bmod 1387 = 996\\
x_{10} &= F(x_9) = (996^2 - 1) \bmod 1387 = 310\\
x_{11} &= F(x_{10}) = (310^2 - 1) \bmod 1387 = 396\\
x_{12} &= F(x_{11}) = (396^2 - 1) \bmod 1387 = 84\\
x_{13} &= F(x_{12}) = (84^2 - 1) \bmod 1387 = 120\\
x_{14} &= F(x_{13}) = (120^2 - 1) \bmod 1387 = 529\\
x_{15} &= F(x_{14}) = (529^2 - 1) \bmod 1387 = 1053\\
x_{16} &= F(x_{15}) = (1053^2 - 1) \bmod 1387 = 595\\
x_{17} &= F(x_{16}) = (595^2 - 1) \bmod 1387 = 339\\
x_{18} &= F(x_{17}) = (339^2 - 1) \bmod 1387 = 1186
\end{aligned}$$

Wie man erkennt, ist

$$x_{18} = x_6$$

Berechnen wir die Iterationen

$$x_i \longleftarrow (x_{i-1}^2 - 1) \bmod 19$$

so erhalten wir folgende Werte:

$$\begin{aligned}
x_1 &\longleftarrow 2\\
x_2 &\longleftarrow (x_1^2 - 1) \bmod 19 \equiv 3 \bmod 19\\
x_3 &\longleftarrow (x_2^2 - 1) \bmod 19 \equiv 8 \bmod 19\\
x_4 &\longleftarrow (x_3^2 - 1) \bmod 19 \equiv 6 \bmod 19\\
x_5 &\longleftarrow (x_4^2 - 1) \bmod 19 \equiv 16 \bmod 19\\
x_6 &\longleftarrow (x_5^2 - 1) \bmod 19 \equiv 8 \bmod 19 = x_3.
\end{aligned}$$

Die Iterationen

$$x_i \longleftarrow (x_{i-1}^2 - 1) \bmod 73$$

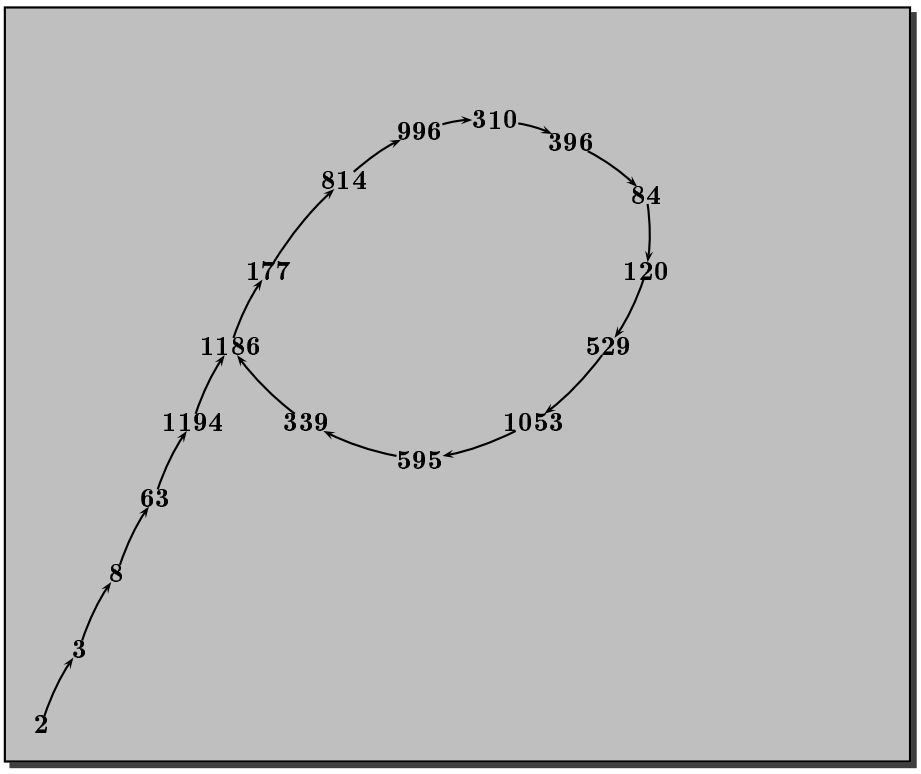


Abbildung 19.2: POLLARDs Rho Heuristik. Die Grafik zeigt die Bahn, die durch die Rekursion $x_{i+1} \longleftarrow (x_i^2 - 1) \bmod 1387$ mit dem Startwert $x_1 = 2$ erzeugt werden.

liefern folgende Werte:

$$
\begin{aligned}
x_1 &\longleftarrow 2\\
x_2 &\longleftarrow (x_1^2 - 1) \bmod 73 \equiv 3 \bmod 73\\
x_3 &\longleftarrow (x_2^2 - 1) \bmod 73 \equiv 8 \bmod 73\\
x_4 &\longleftarrow (x_3^2 - 1) \bmod 73 \equiv 63 \bmod 73\\
x_5 &\longleftarrow (x_4^2 - 1) \bmod 73 \equiv 26 \bmod 73\\
x_6 &\longleftarrow (x_5^2 - 1) \bmod 73 \equiv 18 \bmod 73\\
x_7 &\longleftarrow (x_6^2 - 1) \bmod 73 \equiv 31 \bmod 73\\
x_8 &\longleftarrow (x_7^2 - 1) \bmod 73 \equiv 11 \bmod 73\\
x_9 &\longleftarrow (x_8^2 - 1) \bmod 73 \equiv 47 \bmod 73\\
x_{10} &\longleftarrow (x_9^2 - 1) \bmod 73 \equiv 18 \bmod 73 = x_6.
\end{aligned}
$$

Beispiel 2:

In einem zweiten Beispiel suchen wir die Faktorisierung von $n = 1219$ mit Hilfe

der Methode von POLLARD. Wir setzen also

$$\begin{aligned} n &= 1219 \\ s &= 2 = x_1 = y \\ F(x) &= x^2 - 1 \bmod n \end{aligned}$$

Wir berechnen zunächst die Iterationen. Wir erhalten:

$$\begin{aligned}
x_1 &\longleftarrow 2 \\
x_2 &\longleftarrow (x_1^2 - 1) \bmod 1219 \equiv 3 \bmod 1219 \\
x_3 &\longleftarrow (x_2^2 - 1) \bmod 1219 \equiv 8 \bmod 1219 \\
x_4 &\longleftarrow (x_3^2 - 1) \bmod 1219 \equiv 63 \bmod 1219 \\
x_5 &\longleftarrow (x_4^2 - 1) \bmod 1219 \equiv 311 \bmod 1219 \\
x_6 &\longleftarrow (x_5^2 - 1) \bmod 1219 \equiv 419 \bmod 1219 \\
x_7 &\longleftarrow (x_6^2 - 1) \bmod 1219 \equiv 24 \bmod 1219 \\
x_8 &\longleftarrow (x_7^2 - 1) \bmod 1219 \equiv 575 \bmod 1219 \\
x_9 &\longleftarrow (x_8^2 - 1) \bmod 1219 \equiv 275 \bmod 1219 \\
x_{10} &\longleftarrow (x_9^2 - 1) \bmod 1219 \equiv 46 \bmod 1219 \\
x_{11} &\longleftarrow (x_{10}^2 - 1) \bmod 1219 \equiv 896 \bmod 1219 \\
x_{12} &\longleftarrow (x_{11}^2 - 1) \bmod 1219 \equiv 713 \bmod 1219 \\
x_{13} &\longleftarrow (x_{12}^2 - 1) \bmod 1219 \equiv 45 \bmod 1219 \\
x_{14} &\longleftarrow (x_{13}^2 - 1) \bmod 1219 \equiv 805 \bmod 1219 \\
x_{15} &\longleftarrow (x_{14}^2 - 1) \bmod 1219 \equiv 735 \bmod 1219 \\
x_{16} &\longleftarrow (x_{15}^2 - 1) \bmod 1219 \equiv 207 \bmod 1219 \\
x_{17} &\longleftarrow (x_{16}^2 - 1) \bmod 1219 \equiv 183 \bmod 1219 \\
x_{18} &\longleftarrow (x_{17}^2 - 1) \bmod 1219 \equiv 575 \bmod 1219 = x_8.
\end{aligned}$$

Damit können wir die ggT in den jeweiligen Schritten leicht angeben:

1. $\text{ggT}(8 - 3, 1219) = 1$.
2. $\text{ggT}(311 - 8, 1219) = 1$.
3. $\text{ggT}(24 - 63, 1219) = 1$.
4. $\text{ggT}(275 - 311, 1219) = 1$.
5. $\text{ggT}(896 - 419, 1219) = 53$.

Damit haben wir die Faktorisierung von $n = 1219$ gefunden, denn

$$23 \times 53 = 1219.$$

19.2 Pseudozufallszahlen

Die Sicherheit vieler kryptographischer Systeme hängt entscheidend davon ab, zufällige Größen zu erzeugen (siehe z. B. [190, pp. 147–163] oder [126, Chap. 3], [73]):

- ✘ der Schlüsselstrom in dem One-Time-Pad Verfahren
- ✘ der geheime Schlüssel im DES Algorithmus
- ✘ die Primzahlen im RSA Algorithmus
- ✘ die Primzahlen im Digital Signature Standard (DSS)
- ✘ der private Schlüssel im Digital Signature Algorithm (DSA)
- ✘ und andere

In all diesen Fällen müssen die generierten Zahlen eine geeignete Größe haben und *zufällig* sein. Dabei ist entscheidend, dass die Wahrscheinlichkeit für die Wahl eines Wertes ausreichend klein ist. Dies schließt aus, dass ein Angreifer Vorteile dadurch erlangt, dass er Suchstrategien optimiert, die auf diesen Wahrscheinlichkeiten basieren.

Beispiel:
Der Schlüsselraum des Data Encryption Standards (DES) hat die Große 2^{56}. Wird ein geheimer Schlüssel k mit Hilfe eines echten Zufallsgenerators erzeugt, muß ein Angreifer bei einer Brute-Force Attacke im Durchschnitt 2^{55} mögliche Schlüssel ausprobieren, bis er den korrekten Schlüssel k findet.

Generiert man andererseits einen Schlüssel k dadurch, daß man zunächst eine zufällige 16-Bit Zahl s erzeugt und diese dann mit Hilfe eines komplizierten, aber öffentlich bekannten Verfahrens f auf 56 Bit erweitert, dann benötigt der Angreifer im Durchschnitt lediglich 2^{15} Schlüssel. Dazu wendet der Angreifer die bekannte Funktion f auf jeden möglichen Wert von s an.

Generell ist ein **Zufallszahlengenerator** eine Funktion, die als Ausgabe eine Folge von `0`en und `1`en erzeugt, und zwar derart, daß an jeder Stelle der Folge – basierend auf den zuvor erzeugten Bits – nicht vorhergesagt werden kann, welche Bits folgen werden. Dies klingt einfacher als es tatsächlich ist. Das Problem liegt darin, daß es schwierig ist, mit Hilfe eines Computers echte zufällige Folgen von Bits zu generieren. Dies liegt daran, daß ein Computer ein deterministisches Gerät ist, welches mit Hilfe von Algorithmen arbeitet. Wird daher der gleiche Zufallszahlengenerator zweimal aufgerufen, generiert er die gleiche Folge von Zahlen. Echte Zufallszahlengeneratoren werden eingesetzt, diese sind jedoch kompliziert und sehr teuer. Typischerweise haben echte Zufallsgeneratoren einen Input, der aus zufälligen Ereignissen der physikalischen Welt stammt. Nimmt man beispielsweise die Neutronenstrahlung einer radioaktiven Substanz und zählt die zerfallenen Atome pro Sekunde – z.B. eine gerade Anzahl zerfallener Atome erzeugt eine `0`, eine ungerade eine `1` – so hat man eine echte zufällige `0,1` Folge.

Daher verwenden kryptographische Anwendungen üblicherweise algorithmische Verfahren zur Erzeugung von zufälligen Zahlen. Diese Algorithmen sind deterministisch und daher erzeugen sie eine `0,1` Folge, die statistisch nicht zufällig

ist. Verfügt man jedoch über einen guten Algorithmus, dann besteht die resultierende `0-1` Folge eine Reihe statistischer Testverfahren (siehe [126]). Solche Zahlen nennt man **Pseudozufallszahlen**.

Eine der meist verbreiteten Techniken zur Erzeugung von Pseudozufallszahlen beruht auf einem Algorithmus von D. H. LEHMER, der diesen im Jahre 1949 entwickelte. Das LEHMER-Verfahren ist auch unter dem Begriff *lineare Kongruenzmethode*[5] bekannt. Der LEHMER Algorithmus arbeitet mit vier magischen Zahlen:

m	der Modulus	$m > 0$
a	der Multiplikator	$0 \leq a < m$
c	das Inkrement	$0 \leq c < m$
X_0	der Startwert oder Seed	$0 \leq X_0 < m$

Die Folge von Zufallszahlen X_n erhält man durch die Anwendung der iterativen Gleichung:

$$X_{n+1} = (a \cdot X_n + c) \bmod m$$

Um zu sehen, wie dieser Algorithmus arbeitet, setzen wir: $m = 10$ und $X_0 = a = c = 7$; mit diesen Werten erhält man die Folge

$$7, 6, 9, 0, 7, 6, 9, 0, \ldots$$

Ein anderes Beispiel ist: $m = 32, a = 7, c = 0$ und Startwert $X_0 = 1$. Hier resultiert die Sequenz:

$$1, 7, 17, 23, 1, 7, \ldots$$

Wie diese beiden einfachen Beispiele zeigen, sind die Zahlenfolgen, die durch die Kongruenzmethode erzeugt werden bei manchen (das sind sogar die meisten) Werten von m, a, c und X_0 alles andere als zufällig. Im ersten Beispiel gibt es 10 mögliche Werte von X_i, der Generator erzeugt jedoch nur vier dieser Werte. Im zweiten Beispiel kann X_i die Werte von 0 bis 31 annehmen und auch hier werden nur vier Werte erzeugt. Man sagt, die Folgen haben eine Periode 4.

Ändert man im zweiten Beispiel den Wert des Parameters a auf $a = 5$, dann resultiert die Folge

$$1, 5, 25, 29, 17, 21, 9, 13, 1, 5, \ldots$$

Die Periode dieser Folge verdoppelt sich damit auf 8.

Der Modulus m sollte sehr groß gewählt werden, so dass die Möglichkeit besteht eine lange Reihe verschiedener zufälliger Zahlen zu generieren.[6] Ein allgemein akzeptiertes Kriterium ist, dass m etwa die Größe hat wie die maximal darstellbare nicht–negative ganze Zahl für eine gegebene Computerplattform. Daher hat m typischerweise die Größe von 2^{31}.

Es gibt drei wichtige Kriterien, die ein Zufallszahlengenerator erfüllen muß:

[5] Eine umfassende Diskussion dieses Algorithmus findet man in KNUTH's *The Art of Computer Programming* Vol 2 [126], Chapter 3.

[6] Man möchte beispielsweise nicht nur 10 verschiedene Solitärspiele spielen, sondern möglichst viele verschiedene Kartenverteilungen erhalten.

- ☐ Die Funktion sollte voll-peroidisch sein. Das bedeutet, die Funktion sollte alle Zahlen zwischen 0 und m erzeugen, bevor eine Wiederholung auftritt. Mit anderen Worten, die Periode der resultierenden Sequenz sollte m sein.
- ☐ Die erzeugte Sequenz sollte zufällig aussehen. Die Folge kann natürlich nicht echt zufällig sein, da sie über einen Algorithmus deterministisch erzeugt wird. Es gibt jedoch eine Reihe statistischer Tests, die man benutzt, um eine Aussage über den Grad der Zufälligkeit der erzeugten Folge treffen zu können.
- ☐ Aus praktischen Gründen sollte die Funktion effektiv auf einer 32-Bit Plattform implementierbar sein.

Mit geeigneten Werten für die Parameter a, c und m können alle drei Bedingungen erfüllt werden. Falls m Primzahl ist und $c = 0$, dann ist für bestimmte Werte von a die Periode der resultierenden Folge $m - 1$, die Zahl 0 fehlt. Für 32-Bit Plattformen ist die Zahl $m = 2^{31} - 1$ eine geeignete Primzahl. Damit ist die erzeugende Funktion:

$$X_{n+1} = (aX_n) \bmod (2^{31} - 1)$$

Obwohl es viele Werte für den Parameter a gibt – genauer gesagt mehr als 2 Milliarden – sind nur eine Handvoll geeignet, um alle drei Kriterien zu erfüllen. Ein solcher Wert ist

$$a = 7^5 = 16.807$$

Dieser Wert wurde ursprünglich für den Zufallszahlengenerator auf der IBM S/360 verwendet. Diese Funktion ist weit verbreitet und ausführlich getestet.

Unglücklicherweise sind diese Art von Pseudozufallszahlengeneratoren für die Kryptographie ungeeignet. Das Problem liegt darin, daß ein Angreifer die komplette Folge berechnen kann[7], wenn

- ☞ er weiß, daß der Lineare Kongruenz Algorithmus verwendet wird,
- ☞ er die Parameter kennt (e.g. $a = 7^5, c = 0, m = 2^{31} - 1$),
- ☞ und er eine einzige Zahl der Folge kennt.

Der Grund liegt im deterministischen Charakter des Algorithmus. Selbst wenn dem Angreifer nur bekannt ist, daß die lineare Kongruenzmethode verwendet wird und die Parameter unbekannt sind, genügt es, eine Handvoll Werte der Folge abzufangen – e.g. X_7, X_8, X_9 und X_{10} – um die komplette Folge zu rekonstruieren. Der folgende Satz von Gleichungen

$$\begin{aligned} X_8 &= (aX_7 + c) \bmod m \\ X_9 &= (aX_8 + c) \bmod m \\ X_{10} &= (aX_9 + c) \bmod m \end{aligned}$$

kann nach den Parametern a, c und m aufgelöst werden.

[7] Siehe dazu [193, pp. 369 – 372] und dort zitierte Quellen.

In der Kryptographie gelten deutlich schärfere Anforderungen an Zufallszahlen als bei Computerspielen oder Simulationen. Es darf für einen Angreifer auch bei hohem materiellen Einsatz (i.e. Rechneraufwand) und sehr vielen Versuchen nicht möglich sein, eine Zufallszahl vorherzusagen.

Aus diesem Grund erfordern **kryptographisch sichere Pseudozufallszahlengeneratoren** viel höhere Anforderungen, die in [193, pp. 44-46] auf drei Punkte zusammengefaßt sind:

① Die von einem kryptographisch sicheren Pseudozufallszahlengenerator erzeugte Folge muß zufällig aussehen. Das bedeutet, die Folge muß sämtliche bekannten statistischen Tests bestehen (siehe z.B. [126]).

② Die Folge ist unberechenbar. Es muß rechnerisch unmöglich sein, vorauszusagen, welches das nächste zufällige Bit ist, selbst wenn der Algorithmus bekannt ist und alle zuvor erzeugten Bits des zufälligen Datenstroms.

③ Die Bitfolge kann nicht zuverlässig reproduziert werden. Wendet man den Generator zur Erzeugung der Folge zweimal an mit exakt dem gleichen Input, so erhält man zwei völlig unterschiedliche, unkorrelierte Bitfolgen.

Für kryptographische Anwendungen ist es zwckmäßig, die zur Verfügung stehende Verschlüsselungslogik zu benutzen, um zufällige Bitfolgen zu generieren. Eine Reihe verschiedener Verfahren sind im praktischen Einsatz.

19.2.1 Zyklische Verschlüsselung

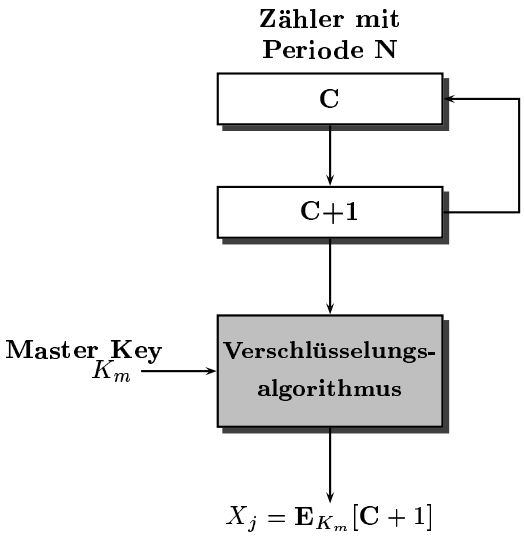


Abbildung 19.3: Pseudozufallszahlengenerator mit Hilfe eines Zählers.

Die Abbildung [19.3] illustriert ein Verfahren, das genutzt wird, um Sitzungsschlüssel aus einem Master-Key zu generieren. Dabei liefert ein Zähler mit Peroide N den Input in einen Verschlüsselungsalgorithmus. Sollen beispielsweise 56-Bit DES Schlüssel generiert werden, kann ein Zähler mit Periode 2^{56} benutzt werden. Nach der Erzeugung jedes einzelnen Keys wird der Zähler um 1 hochgesetzt. Daher laufen die mit diesem Verfahren generierten Pseudozufallszahlen X_i durch einen vollen Zyklus der Länge N. Jede Zahl $X_0, X_1, \ldots X_{N-1}$ basiert auf einem unterschiedlichen Zählerwert, daher ist

$$X_0 \neq X_1 \neq X_2 \cdots \neq X_{N-1}$$

Da der Master KEy K_m geheim gehalten wird, ist es rechnerisch nicht möglich, aus der Kenntnis eines Wertes X_i oder mehrerer solcher Werte den Session Key X_{i+1} abzuleiten.

19.2.2 ANSI X9.17 Psudozufallszahlengenerator

Einer der stärksten – i.e. kryptographisch sichersten – Pseudozufallszahlengeneratoren ist in der Spezifikation ANSI X9.17 dokumentiert. Eine Reihe von Anwendungen nutzen diesen Generator, wie sichere Finanztransaktionen oder PGP.

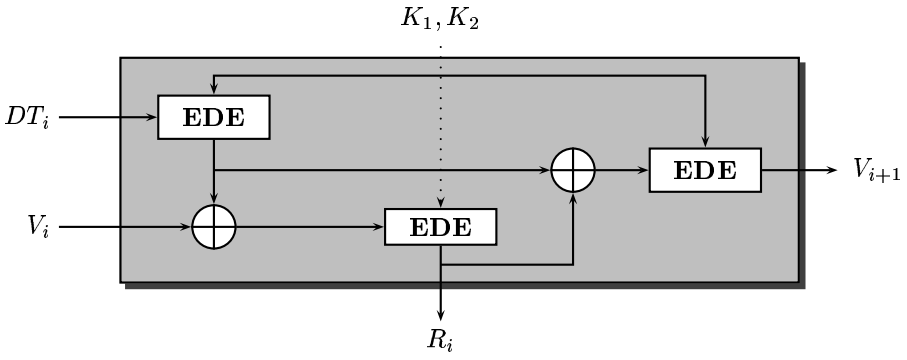


Abbildung 19.4: ANSI X9.17 Pseudozufallszahlengenerator.

Die Abbildung [19.4] illustriert den Algorithmus. Dieser verwendet Triple DES zur Verschlüsselung (i.e. die Kästen mit EDE). Weiterhin werden benutzt:

- ✘ **Input**
  Zwei pseudozufällige Größen steuern den Generator. Eine Größe ist eine 64 Bit Darstellung der aktuelle Uhrzeit und des aktuellen Datums (DT_i), dieser Input wird bei jeder Zufallszahlerzeugung neu belegt. Die andere Größe ist ein 64 Bit Startwert (seed, V_i); dieser Startwert wird durch eine beliebige Zahl initialisiert und bei jeder Erzeugung einer Zahl neu belegt.

✘ **Schlüssel**
Der ANSI Generator benutzt drei Triple DES Verschlüsselungsmodule. Sämtliche drei Module benutzen das gleiche Schlüsselpaar K_1, K_2 (56 Bit Schlüssel). Diese müssen geheim gehalten werden und werden ausschließlich für die Erzeugung von Psudozufallszahlen verwendet.

✘ **Output**
Die Ausgabe ist eine 64 Bit Pseudozufallszahl (R_i) und einen 64 Bit Seed (V_{i+1}).

Definiere die folgenden Größen:

DT_i	Datum/Uhrzeit zu Beginn des $i$ten Iterationsschrittes
V_i	Startwert zu Beginn des $i$ten Iterationsschrittes
R_i	Pseudozufallszahl, die durch den $i$ten Iterationsschritt erzeugt wird
K_1, K_2	DES Schlüssel, die für jeden Schritt verwendet werden

Damit berechnet der Algorithmus in jedem Iterationsschritt die Pseudozufallszahl gemäß:

$$R_i = \texttt{EDE}_{K_1,K_2}\left[V_i \oplus \texttt{EDE}_{K_1,K_2}[DT_i]\right]$$
$$V_{i+1} = \texttt{EDE}_{K_1,K_2}\left[R_i \oplus \texttt{EDE}_{K_1,K_2}[DT_i]\right]$$

wobei EDE für die Triple-DES Sequenz *Encryption–Decryption–Encryption* steht mit zwei verschiedenen Schlüsseln.

Eine Reihe von Faktoren tragen zur kryptographischen Stärke dieses Verfahrens bei. Diese Methode benutzt einen 112 Bit Schlüssel und drei EDE Verschlüsselungen für insgesamt neun DES Verschlüsselungen. Das Verfahren hat als Input zwei psudozufällige Parameter, Uhrzeit und Datum sowie einen Seedwert, der von dem Generator selbst produziert wird und unterschiedlich zu der Pseudozufallszahl ist. Mit anderen Worten, die Menge an Informationen, die ein Angreifer benötigt, um die Pseudozufallszahlen zu erhalten, ist überwältigend. Selbst wenn eine der Pseudozufallszahlen R_i kompromittiert ist, ist es nicht möglich, aus dem bekannten R_i auf den Seedwert V_{i+1} zu schließen, da eine zusätzliche EDE Operation verwendet wird, um den Seedwert V_{i+1} zu erhalten.

19.2.3 Blum-Blum-Shub Pseudozufallszahlengenerator

Der **Blum-Blum-Shub Pseudorandom Bit Generator** (siehe [26] oder [205, pp. 155]) – auch bekannt als x^2 mod n Generator oder BBS Generator – ist ein kryptographisch sicherer Pseudorandom Bit Generator unter der Annahme, daß die Faktorisierung ein hartes Problem ist.

Algorithmus BBS Generator

Der folgende Algorithmus generiert eine pseudozufällige Bitfolge $z_1, z_2, \ldots, z_l$ der Länge l:

BBS0 Erzeuge zwei zufällige große unterschiedliche Primzahlen p und q, jede kongruent 3 modulo 4; berechne $n = pq$.

BBS1 Wähle eine zufällige ganze Zahl s – den Startwert oder *seed* – im Intervall $[1, n-1]$, so daß s und n coprim sind, i.e. $\mathtt{ggT}(s, n)$=1.

BBS2 Berechne $x_0 \leftarrow s^2 \bmod n$.

BBS3 Für i von 1 bis l führe folgende Schritte aus:

3-1 $x_i \leftarrow x_{i-1}^2 \bmod n$

3-2 $z_i \leftarrow$ das niedrigwertigste Bit von x_i

BBS4 Die Ausgabe ist die Folge $z_1, z_2, \ldots, z_l$.

Beispiel:

Um zu sehen, wie der BBS Algorithmus arbeitet, führen wir ein Beispiel detailliert aus. Wir beginnen mit den beiden Primzahlen

$$p = 191$$
$$q = 167$$

Beide Primzahlen sind der Tat kongruent 3 mod 4:

$$p \bmod 4 = 191 \bmod 4 \equiv 3 \bmod 4$$
$$q \bmod 4 = 167 \bmod 4 \equiv 3 \bmod 4$$

Dann ist:

$$n = p \times q = 191 \times 167 = 31.897$$

Als Seed wählen wir $s = 599$ mit

$$\mathtt{ggT}(s, n) = 1$$

Dann folgt:

$$x_0 = s^2 \bmod n = 358.801 \bmod 31.897 = 7.934$$
$$b_0 = 7.934 \bmod 2 = 0$$

Die nächste Runde ergibt:

$$x_1 = (x_0)^2 \bmod n = 62.948.356 \bmod 31.897 = 15.575$$
$$b_1 = 15.575 \bmod 2 = 1$$

$$x_2 = (x_1)^2 \bmod n = 242.580.625 \bmod 31.897 = 3.940$$
$$b_2 = 3.940 \bmod 2 = 0$$

$$x_3 = (x_2)^2 \bmod n = 15.523.600 \bmod 31.897 = 21.658$$
$$b_3 = 21.658 \bmod 2 = 0$$

$$x_4 = (x_3)^2 \bmod n = 469.068.964 \bmod 31.897 = 23.579$$
$$b_4 = 23.579 \bmod 2 = 1$$

$$x_5 = (x_4)^2 \bmod n = 555.969.241 \bmod 31.897 = 4.531$$
$$b_5 = 4.531 \bmod 2 = 1$$

$$x_6 = (x_5)^2 \bmod n = 20.529.961 \bmod 31.897 = 20.190$$
$$b_6 = 20.190 \bmod 2 = 0$$

$$x_7 = (x_6)^2 \bmod n = 407.636.100 \bmod 31.897 = 24.337$$
$$b_7 = 24.337 \bmod 2 = 1$$

$$x_8 = (x_7)^2 \bmod n = 59.2289.569 \bmod 31.897 = 26.073$$
$$b_8 = 26.073 \bmod 2 = 1$$

usw. Mit den gewählen Parametern wird also die folgende Bitsequenz generiert:

```
0 1 0 0 1 1 0 1 1
```

Ein C-Programm, das den BBS Generator mit den obigen Parametern implementiert, liefert für $l = 100$ die folgende Sequenz:

```
n = 31897; x[0] ist 7934
  x[1] ist   15575 und  b[1] ist 1
  x[2] ist    3940 und  b[2] ist 0
  x[3] ist   21658 und  b[3] ist 0
  x[4] ist   23579 und  b[4] ist 1
  x[5] ist    4531 und  b[5] ist 1
  x[6] ist   20190 und  b[6] ist 0
  x[7] ist   24337 und  b[7] ist 1
  x[8] ist   26073 und  b[8] ist 1
  x[9] ist   12465 und  b[9] ist 1
 x[10] ist    5938 und b[10] ist 0
 x[11] ist   13659 und b[11] ist 1
 x[12] ist    2728 und b[12] ist 0
 x[13] ist    9983 und b[13] ist 1
 x[14] ist   14061 und b[14] ist 1
 x[15] ist   14115 und b[15] ist 1
 x[16] ist    4563 und b[16] ist 1
```

```
x[17] ist   24125 und b[17] ist 1
x[18] ist   22963 und b[18] ist 1
x[19] ist   10062 und b[19] ist 0
x[20] ist    2766 und b[20] ist 0
x[21] ist   27373 und b[21] ist 1
x[22] ist   20599 und b[22] ist 1
x[23] ist   24907 und b[23] ist 1
x[24] ist   25793 und b[24] ist 1
x[25] ist    3120 und b[25] ist 0
x[26] ist    5815 und b[26] ist 1
x[27] ist    3405 und b[27] ist 1
x[28] ist   15414 und b[28] ist 0
x[29] ist   22540 und b[29] ist 0
x[30] ist   28081 und b[30] ist 1
x[31] ist   16824 und b[31] ist 0
x[32] ist   24895 und b[32] ist 1
x[33] ist    2315 und b[33] ist 1
x[34] ist     529 und b[34] ist 1
x[35] ist   24665 und b[35] ist 1
x[36] ist   22641 und b[36] ist 1
x[37] ist   30091 und b[37] ist 1
x[38] ist    8142 und b[38] ist 0
x[39] ist   10198 und b[39] ist 0
x[40] ist   14984 und b[40] ist 0
x[41] ist   29170 und b[41] ist 0
x[42] ist    4528 und b[42] ist 0
x[43] ist   24910 und b[43] ist 0
x[44] ist   15759 und b[44] ist 1
x[45] ist   27936 und b[45] ist 0
x[46] ist   28094 und b[46] ist 0
x[47] ist   13468 und b[47] ist 0
x[48] ist   20682 und b[48] ist 0
x[49] ist    6354 und b[49] ist 0
x[50] ist   23611 und b[50] ist 1
x[51] ist   15452 und b[51] ist 0
x[52] ist   15259 und b[52] ist 1
x[53] ist   20878 und b[53] ist 0
x[54] ist   18379 und b[54] ist 1
x[55] ist   30308 und b[55] ist 0
x[56] ist    5058 und b[56] ist 0
x[57] ist    1970 und b[57] ist 0
x[58] ist   21363 und b[58] ist 1
x[59] ist   27390 und b[59] ist 0
x[60] ist   26557 und b[60] ist 1
x[61] ist   31579 und b[61] ist 1
x[62] ist    5433 und b[62] ist 1
x[63] ist   12764 und b[63] ist 0
x[64] ist   21717 und b[64] ist 1
x[65] ist   30944 und b[65] ist 0
x[66] ist   15093 und b[66] ist 1
```

```
x[67] ist   22172 und b[67] ist 0
x[68] ist    1020 und b[68] ist 0
x[69] ist   19696 und b[69] ist 0
x[70] ist    1102 und b[70] ist 0
x[71] ist    2318 und b[71] ist 0
x[72] ist   14428 und b[72] ist 0
x[73] ist    7362 und b[73] ist 0
x[74] ist    6041 und b[74] ist 1
x[75] ist    3513 und b[75] ist 1
x[76] ist   28927 und b[76] ist 1
x[77] ist   17328 und b[77] ist 0
x[78] ist   13123 und b[78] ist 1
x[79] ist    1226 und b[79] ist 0
x[80] ist    3917 und b[80] ist 1
x[81] ist     432 und b[81] ist 0
x[82] ist   27139 und b[82] ist 1
x[83] ist   23591 und b[83] ist 1
x[84] ist   28322 und b[84] ist 0
x[85] ist   21825 und b[85] ist 1
x[86] ist   12724 und b[86] ist 0
x[87] ist   22901 und b[87] ist 1
x[88] ist    5327 und b[88] ist 1
x[89] ist   20496 und b[89] ist 0
x[90] ist    2526 und b[90] ist 0
x[91] ist    1276 und b[91] ist 0
x[92] ist    1429 und b[92] ist 1
x[93] ist     633 und b[93] ist 1
x[94] ist   17925 und b[94] ist 1
x[95] ist    7144 und b[95] ist 0
x[96] ist    1536 und b[96] ist 0
x[97] ist   30815 und b[97] ist 1
x[98] ist   22432 und b[98] ist 0
x[99] ist   19449 und b[99] ist 1
```

Dies führt auf die Bit-Sequenz:

```
0 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0
0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1 1 1 0 0
0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0
1 1 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 1 0
1 0 1 1 0 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1
```

Ein anderes Beispiel ist in der folgenden Tabelle aufgelistet ([205]):

Step	x_i	b_i		**Step**	x_i	b_i
0	20749			11	137922	0
1	143135	1		12	123175	1
2	177671	1		13	8630	0
3	97048	0		14	114386	0
4	89992	0		15	14863	1
5	174051	1		16	133015	1
6	80649	1		17	106065	1
7	45663	1		18	45870	0
8	69442	0		19	137171	1
9	186894	0		20	48060	0
10	177046	0				

In diesem Beispiel sind die Parameter $n = 192649 = 383 \times 503$ mit Seed $s = 101.355$.

19.3 Weiteres über Primzahlen

Eine Reihe von Public-Key Verschlüsselungssystemen – beispielsweise RSA – erzeugen in einem ersten Schritt Primzahlen. Im RSA Verfahren werden diese beiden Primzahlen multipliziert, die daraus entstehende Zahl ist Teil des Public Keys. Diese Primzahlen haben eine bestimmte Größe, typischerweise 512 Bits. In diesem Abschnitt sehen wir uns zwei Aspekte an, die mit Primzahlen zusammenhängen:

① Gibt es genug Primzahlen dieser Größe – oder genauer, wieviele Primzahlen gibt es überhaupt bis zu einer gegebenen Zahl x?

② Wie kann man erkennen, daß eine Folge von 512 Bits – die in der Regel auch noch zufällig sein soll – eine Primzahl codiert?

In diesem Kapitel untersuchen wir eine Reihe von Eigenschaften von Primzahlen, die für die Kryptographie relevant sind. Weiterführende Diskussionen findet man in [34, 52, 172] oder [179].

19.3.1 Umrechung großer Zahlen

In der Kryptographie ist man häufig mit der Frage konfrontiert, daß eine bestimmte Anzahl von Bits gegeben sind – beispielsweise 512 Bit für die Primzahlen der RSA Verschlüsselung oder 56 Bit für den DES Schlüssel – und gesucht ist die Größenordnung der dadurch codierten Zahl in Dezimalschreibweise, oder anders ausgedrückt, gesucht ist die Anzahl der Stellen dieser Zahl (siehe z.B. [146]).

Zunächst folgt aus der Binärcodierung, daß mit N Bit natürliche Zahlen im Intervall $[0, 2^N - 1]$ codiert werden. Da nur die Größenordnung von Interesse ist, gehen wir im folgenden davon aus, daß durch N Bits die Zahl 2^N codiert wird. Für 100 Bit, $N = 100$ erhält man daher die Zahl 2^{100}. Dies ist sicherlich eine große Zahl, aber wie groß ist sie wirklich?

Faustregel

Wir wissen, daß

$$2^3 = 8 < 10$$

Nimmt man beide Seiten dieser Ungleichung zur 33. Potenz folgt:

$$2^{99} < 10^{33}$$

Somit erhält man

$$2^{100} < 2 \cdot 10^{33}$$

Nun ist $2 \cdot 10^{33}$ eine 2 gefolgt von 33 Nullen, i.e., diese Zahl hat 34 Stellen. Mit dieser Abschätzung erhalten wir daher das Ergebnis, daß 2^{100} höchstens 34 Stellen hat (obere Schranke).

Wir können eine untere Grenze für die Größe der Zahl auf folgende Weise abschätzen. Da

$$2^{10} = 1024 > 1000 = 10^3$$

folgt:

$$2^{100} > 10^{30}$$

i.e. die Zahl 2^{100} hat mindestens 31 Stellen.

Exakte Ableitung

Die Abschätzung der Größe der Zahl kann genauer durchgeführt werden. Die Tatsache, daß eine Zahl genau k Stellen hat, bedeutet, daß die Zahl zwischen 10^{k-1} und 10^k liegt, wobei die untere Schranke angenommen werden darf, die obere jedoch nicht[8]. Gesucht ist also der Wert k mit:

$$10^{k-1} \le 2^{100} < 10^k$$

Logarithmiert man diese Ungleichung (zur Basis 10), erhält man:

$$k - 1 \le \log_{10} 2^{100} = 100 \cdot \log_{10} 2 < k$$

Das bedeutet, $k-1$ ist die größte ganze Zahl kleiner oder gleich $100 \cdot \log_{10} 2$. Für diesen Sachverhalt verwendet man die GAUSS Klammer:

$$k - 1 = \lfloor 100 \cdot \log_{10} 2 \rfloor$$

die den ganzzahligen Anteil der reellen Zahl $100 \cdot \log_{10} 2$ beschreibt.

Mit Hilfe eines Taschenrechners findet man leicht für das konkrete Zahlenbeispiel:

$$\log_{10} 2 = 0.30102$$

womit folgt

$$100 \cdot \log_{10} 2 \approx 30.102$$

und

$$\lfloor 100 \cdot \log_{10} 2 \rfloor = 30 = k - 1$$

Damit hat die Zahl 2^{100} *genau* 31 Stellen.

Allgemein gilt die folgende Aussage: Eine Zahl der Form 2^N hat genau

$$k = 1 + \lfloor N \cdot \log_{10} 2 \rfloor$$

Stellen.

[8] Dieses Argument wird anhand eines Beispiels klar: Sei $z = 49348$, i.e. die Zahl hat genau $k = 5$ Stellen. Dann ist

$$10^{k-1} = 10\,000 \le z = 49348 < 10^5 = 100\,000$$

19.3.2 Der Primzahlsatz

Die Mathematiker befassen sich bereits seit der Antike mit der Frage, wieviele Primzahlen es überhaupt gibt. Es ist seit über 2 000 Jahren bekannt – dies wurde von EUKLID gezeigt (siehe Abschnitt [14.2]) – dass es unendlich viele Primzahlen gibt. Andererseits ist kein Verfahren bekannt, das systematisch alle Primzahlen erzeugt. Es gibt jedoch sehr mächtige Theoreme der Zahlentheorie, die statistische Aussagen über die Verteilung von Primzahlen liefern[9].

Falls wie bei der RSA Verschlüsselung 512 Bits für eine Primzahl zur Verfügung stehen, dann folgt aus dem vorigen Abschnitt, dass damit Zahlen zwischen

$$0 \text{ bis } 2^{512} \sim 10^{154}$$

dargestellt werden können.

Check:

Nach den im vorigen Abschnitt abgeleiteten Zusammenhängen haben wir: $k =$ Anzahl der Dezimalstellen

$$k = 1 + \lfloor N \cdot \log_{10} 2 \rfloor = 1 + \lfloor 512 \cdot 0.30102 \rfloor = 1 + \lfloor 154.12224 \rfloor = 155$$

Die Frage ist nun:

Wieviele Primzahlen gibt es, die kleiner sind als die Zahl $x = 10^{154}$*?*

Die Antwort darauf gibt der sogenannte **Primzahlsatz**. Die Anzahl der Primzahlen bis zu einer gewählten Zahl x ist durch den Integral Logarithmus gegeben:

$$\pi(x) = \int_2^x \frac{1}{\ln y}\, dy$$

Gemäß Arbeiten von CARL FRIEDRICH GAUSS und LEGENDRE ist dieses Integral approximierbar durch:

$$\pi(x) \approx \frac{x}{\ln x}$$

Eine Konsequenz dieser Aussage ist, daß die Wahrscheinlichkeit dafür, daß eine zufällig gewählte Zahl x eine Primzahl ist, durch $\frac{1}{\ln x}$ gegeben ist. Mit Hilfe des Primzahlsatzes gewinnen wir daher die Erkenntnis, falls wir Zahlen der Länge von 512 Bits betrachten, haben wir etwa 10^{150} Primzahlen zur Verfügung.

Anmerkungen:

Eine Konsequenz aus dem Primzahlsatz ist folgende. Gegeben ist eine Zahl $n \sim 10^k$ mit k *Dezimalstellen*, die mit der Teilermethode faktorisiert werden soll. Wenn n zusammengesetzt ist, dann hat diese Zahl einen Teiler im Intervall $[2, \lfloor\sqrt{n}\rfloor]$, i.e. der Teiler ist kleiner oder gleich $10^{k/2}$. Bis zu der Zahl $10^{k/2}$ gibt es nach dem Primzahlsatz etwa

$$\frac{10^{k/2}}{k/2 \cdot \ln 10}$$

[9] Siehe e.g. [195], Chapter 4 oder die Prime Page mit URL:

`http://www.utm.edu/research/primes`

Primzahlen, also müssen im worst case auch ebenso viele Probedivisionen bei der Teilermethode durchgeführt werden.

Unter der Annahme, dass man pro Sekunde 10^6 Divisionen auf einem Rechner durchführen kann, benötigt ein Primzahltest, der auf der Teilermethode beruht für das Testen einer Zahl mit k Dezimalstellen etwa

$$\frac{10^{k/2}}{(k/2)\cdot \ln 10 \cdot 10^6 \cdot 60 \cdot 60 \cdot 24 \cdot 365} = \frac{10^{k/2}}{k} \cdot 2.75 \cdot 10^{-14} \text{ Jahre.}$$

Beispiele:

$$\begin{array}{lcl} k = 30 & \Longrightarrow & 11 \text{ Monate} \\ k = 50 & \Longrightarrow & 5.5 \cdot 10^9 \text{ Jahre} \\ k = 100 & \Longrightarrow & 2.75 \cdot 10^{34} \text{ Jahre} \end{array}$$

Dieses Verhalten drückt in einer anderen Weise aus, dass es ein 'hartes' Problem ist, ganze Zahlen zu faktorisieren.

Hier sind einige Antworten auf typische Fragen im Umfeld von Primzahlen (siehe [193, p. 258]):

❶ Falls jeder Internetbenutzer – das sind mittlerweile einige Millionen – eine unterschiedliche Primzahl benötigt, reichen diese 10^{150} Primzahlen aus?

Die Antwort ist ein entscheidendes **Ja**. Um ein Gefühl für große Zahlen zu erhalten, kann man folgende Überlegung anstellen. Im Universum gibt es ungefähr 10^{80} Atome. Falls jedes dieser Atome 1 Milliarde (10^9) Primzahlen jede Mikrosekunde (10^{-6}) seit dem Urknall benötigt – das sind mal gerade 10^9 Jahre her – benötigt man etwa 10^{111} Primzahlen[10]. Vergleicht man diese 10^{111} mit 10^{150}, dann ist dies eine zu vernachlässigende Zahl

❷ Was passiert, wenn zwei Personen zufällig die gleiche Primzahl wählen?

Die Wahrscheinlichkeit dafür ist verschwindend gering, dies wird nicht passieren!

❸ Eine häufig gestellte Frage ist, ob man nicht eine Datenbank erstellen kann, die ein für allemal sämtliche Primzahlen bis 2^{512} enthält. Ein Angreifer

[10] Die Rechnung geht so:

(a) 1 Jahr hat

$$60 \cdot 60 \cdot 24 \cdot 365 = 31\,536\,000$$

Sekunden.

(b) Seit dem Urknall sind dann

$$10^9 \cdot 31\,536\,000 \cdot 10^6 = 3.1 \cdot 10^{22}$$

Mikrosekunden vergangen.

(c) In jeder Mikrosekunde werden 10^9 Primzahlen benötigt, das sind daher

$$3.1 \cdot 10^{22} \cdot 10^9 \approx 10^{31}$$

Primzahlen.

(d) Dieser Wert mit den 10^{80} Atomen multipliziert, ergibt 10^{111} Primzahlen.

kann diese Datenbank dann benuzten, um die Public-Key Algorithmen zu knacken.

Die Antwort auf diese Frage ist Radio-Eriwan-Ähnlich: Im Prinzip ja, der Angreifer hat aber nichts von seiner Primzahlsammlung. Falls man die Informationsmenge von 1 GB Kapazität auf einer Festplatte speichern kann, die 1 Gramm wiegt, dann wiegt die Liste der 512-Bit Primzahlen so viel, daß die CHANDRASEKHAR Grenze überschritten wird und das Speichermedium in eine Schwarzes Loch kollabiert – daher kann der Angreifer nicht mehr auf die Primzahlsammlung zugreifen.

Die Rechnung geht folgendermaßen:

- ✓ Eine Primzahl wird durch 512 Bit dargestellt. Das bedeutet, jede Primzahl benötigt 512 Bit = 64 Byte Speicherplatz.
- ✓ 1 KB Speicherplatz wird benötigt, um 16 Primzahlen zu speichern.
- ✓ Auf 1 MB Speicherplatz können dann etwa $16\,000 = 16 \cdot 10^3$ Primzahlen gespeichert werden.
- ✓ Auf 1 GB Speicherplatz können dann etwa $16 \cdot 10^6$ Primzahlen gespeichert werden.
- ✓ Für 10^{150} Primzahlen abzuspeichern benötigt man etwa 10^{142} Platten mit der Kapazität von 1 GB.
- ✓ Da eine GB Platte 1 Gramm wiegt, wiegt der Speicherschrank mit den 10^{142} Platten etwa 10^{136} Tonnen.
- ✓ Die Masse der Sonne ist etwa $2 \cdot 10^{27}$ Tonnen, damit hat unser Speicherschrank die Masse von etwa 10^{109} Sonnen, das produziert das supermassivste Schwarze Loch überhaupt.

19.3.3 Primzahltests

Da es offenbar keinen Mangel an Primzahlen gibt, stellt sich nun die Frage, wie man eine solche Primzahl überhaupt findet[11]. Da – wie mehrfach erwähnt und wie oben gezeigt – die Faktorisierung ein hartes Problem ist, stellt sich die Frage, wie die Erzeugung von Primzahlen effektiv durchgeführt werden kann. Der entscheidende Punkt ist, dass das Entscheidungsproblem

Ist N eine Primzahl?

viel einfacher zu beantworten ist – nämlich mit Ja oder Nein – als die kompliziertere Frage:

Was sind die Primfaktoren von N?

Der Holzweg, um Primzahlen zu finden ist die Erzeugung einer zufälligen Bitfolge und anschließend wird versucht, diese Zahl zu faktorisieren. Der effektive

[11] Siehe z.B. [172].

Abbildung 19.5: Spezialcomputer zur Lösung zahlentheoretischer Probleme, bestehend aus Sägeblock, Fahrradketten und einem Motor. Mit diesem Rechner können Primzahltests durchgefÜhrt werden. Diese Rechenmaschine läßt sich mechanisch programmieren, indem kleine Stifte in Fahrradketten gesteckt werden. Der Fahrradkettencomputer ist ein Nachbau einer Rechenmaschine, die D.H. LEHMER im Jahre 1926 an der UCLA entwickelte [172].

Weg besteht darin, eine zufällige Bitfolge zu generieren und anschließend zu **testen**, ob diese Bitfolge eine Primzahl codiert oder nicht.

Es gibt effektivere Primzahltest als die Teilermethode:

- der effektivste deterministische Primzahltest von ADLEMAN et al. hat eine Komplexität von

$$O\Big((\log n)^{c\cdot\log n\cdot\log\log n}\Big)$$

- ein determinister Test von MILLER hat ein Laufzeitverhalten von

$$O\big((\log n)^5\big)$$

- probabalistische Primzahltests e.g. von MILLER-RABIN oder SOLOVAY-STRASSEN sind hingegen sehr effizient mit einem Laufzeitverhalten

$$O\big(\log\frac{1}{\epsilon}\cdot(\log n)^3\big)$$

  mit Fehlerwahrscheinlichkeit ϵ.

Zahlenbeispiele für den MILLER-RABIN Test:

Für k-stellige Dezimalzahlen mit Fehlerwahrscheinlichkeit von $\epsilon = 10^{-100}$ bei 10^6 Operationen pro Sekunde:

- $k = 30 \implies$ 1 sec.
- $k = 50 \implies$ 12.5 sec.
- $k = 100 \implies$ 100 sec.

Seit August 2002 ist durch eine Arbeit der drei indischen Mathematiker MANINDRA AGRAWAL, NEERAJ KAYAL und NITIN SAXENA gezeigt [1], dass das Entscheidungsproblem der Erkennung von Primzahlen effizient zu lösen ist, i.e. das Entscheidungsproblem PRIMES liegt in der Komplexitätsklasse $\mathcal{P}$.

Es gibt eine Reihe probabilistischer Primzahltests, die typischerweise folgendermaßen arbeiten:

> Entweder ist die getestete Zahl sicher keine Primzahl oder die Zahl ist mit einer bestimmten Wahrscheinlichkeit eine Primzahl.

Zu diesen Tests zählen

- Der **Solovay-Strassen Test**
- Der **Lehmann Test**
- Der **Miller-Rabin Test**

Solovay-Strassen Test

Der SOLOVAY-STRASSEN Test beruht letztendlich auf den kleinen FERMATschen Theorem in der Form:

$$N \text{ prim} \implies a^{N-1} \equiv 1 \bmod N \quad \forall\, a \in \mathbb{Z}_N \setminus \{0\}$$

Die Idee des Testens beruht also auf folgender Vorgehensweise: Man wählt mit einem Pseudozufallszahlengenerator eine Reihe von Zahlen $a \in \mathbb{Z}_N \setminus \{0\}$ aus und berechnet deren $N-1$-te Potenzen. Falls eine einzige Potenz $\neq 1$ wird, kann aus dem kleinen FERMATschen Theorem gefolgert werden, dass die Zahl N nicht prim ist. Falls jedoch alle berechneten Potenzen den Wert 1 ergeben, kann nur gefolgert werden, dass N wahrscheinlich prim ist.

Der Grund dafür, dass nur mit einer bestimmten Wahrscheinlichkeit gefolgert werden kann, dass N eine Primzahl ist, liegt darin, dass es Zahlen gibt, die einerseits das FERMAT-Kriterium erfüllen, aber andererseits nicht prim sind. Also das Argument ist:

> Wenn N eine Primzahl ist, dann gilt $a^{N-1} \equiv 1 \bmod N$

Die Umkehrung gilt jedoch nicht[12], i.e. es gibt Zahlen, die die FERMATsche Eigenschaft besitzen, aber keine Primzahlen sind. Solche Zahlen nennt man **Pseudo-Primzahlen** (siehe [52, pp. 120]).

[12] In [59, p. 19] ist dieser Sachverhalt mit einem griffigen Beispiel illustriert: Alle Autos haben Räder, oder anders formuliert, wenn ein Ding ein Auto ist, dann folgt, dass es Räder hat. Umgekehrt muss die Aussage nicht wahr sein, d.h. wenn ein Ding Räder hat, dann ist es nicht notwendigerweise ein Auto. Fahrräder oder Inline-Skater haben auch Räder.

Beispiel:

Die kleinste Pseudo-Primzahl ist $q = 341$. Es gilt die Faktorisierung $341 = 11 \cdot 31$ aber

$$2^{340} - 1 \equiv 0 \bmod 341$$

denn mit dem `fastexp`-Algorithmus (siehe Kapitel [15.3.3]) überzeugt man sich leicht davon, dass $q = 341$ das FERMAT-Kriterium zur Basis 2 erfüllt:

$$340 = 256 + 64 + 16 + 4$$

und

$$\begin{aligned}
2^2 \bmod 341 &\equiv 4 \bmod 341 \\
2^4 \bmod 341 &\equiv 16 \bmod 341 \\
2^8 \bmod 341 &\equiv 256 \bmod 341 \\
2^{16} \bmod 341 &\equiv 64 \bmod 341 \\
2^{32} \bmod 341 &\equiv 4 \bmod 341 \\
2^{64} \bmod 341 &\equiv 16 \bmod 341 \\
2^{128} \bmod 341 &\equiv 256 \bmod 341 \\
2^{256} \bmod 341 &\equiv 64 \bmod 341
\end{aligned}$$

Damit ist

$$2^{340} \bmod 341 = (64 \cdot 16 \cdot 64 \cdot 16) \bmod 341 \equiv 1 \bmod 341,$$

daher ist $2^{340} - 1$ ohne Rest durch 341 teilbar.

Weitere Pseudo-Primzahlen sind: 561,1105,1387 und 2047.

Carmichael Zahlen

Es gibt Pseudo-Primzahlen n, die den FERMAT-Test

$$a^{n-1} \equiv 1 \bmod n$$

für alle Basen a, die coprim zu n sind – i.e. $\texttt{ggT}(a, n)$ bestehen. obwohl die zu testende Tahl n nicht prim ist. Diese Zahlen nennt man **Carmichael Zahlen**. Die kleinste CARMICHAEL Zahl ist

$$n = 561.$$

Wenn nun basierend auf dem kleinen FERMATschen Satz getestet wird, ob eine Zahl N prim ist und dabei resultiert, dass $2^{N-1} - 1$ tatsächlich durch N teilbar ist, kann nur gefolgert werden, dass N eine Primzahl oder eine Pseudo-Primzahl ist. Allerdings spricht die Wahrscheinlichkeit für die Primzahl-Eigenschaft, denn obwohl es unendlich viele Pseudo-Primzahlen gibt, sind sie sehr selten. Betrachtet man die natürlichen Zahlen von 1 bis 1.000, so gibt es nur zwei Pseudo-Primzahlen, nämlich 341 und 561. Bei den natürlichen Zahlen bis 10^6 gibt es 245 Pseudo-Primzahlen.

Monte Carlo Algorithmus Solovay Strassen

S0 Input: Zahl N

S1 Wähle k zufällige Zahlen $a_1, a_2, \ldots, a_k \in \mathbb{Z}_N \setminus \{0\}$ (typischerweise $k \approx 50$)

S2 Wiederhole die folgenden Anweisungen für $i = 1, 2, \ldots, k$

S2.1 Berechne $a_i^{N-1} \bmod N$

S2.2 Falls $a_i^{N-1} \bmod N \not\equiv 1 \bmod N$, STOP; Return: N nicht prim

S3 Falls in [S2] immer $a_i^{N-1} \bmod N \equiv 1 \bmod N$ resultiert STOP; Return: N ist wahrscheinlich prim.

Beispiele:

Sei $N = 97$; es soll mit Hilfe des SOLOVAY-STRASSEN Tests geprüft werden, ob N prim ist oder nicht. Zunächst zerlegt man: $N - 1 = 96 = 64 + 32$. Dann wählen wir eine Zufallszahl in $\mathbb{Z}_{97} \setminus \{0\}$, etwa 7.

Dann berechnen wir:

$$a^{N-1} \bmod N = 7^{96} \bmod 97$$

Wir wenden den `fastexp`-Algorithmus aus Abschnitt [15.3.3] an und erhalten:

$$\begin{aligned} 7^2 \bmod 97 &\equiv 49 \bmod 97 \\ 7^4 \bmod 97 &\equiv 75 \bmod 97 \\ 7^8 \bmod 97 &\equiv 91 \bmod 97 \\ 7^{16} \bmod 97 &\equiv 36 \bmod 97 \\ 7^{32} \bmod 97 &\equiv 35 \bmod 97 \\ 7^{64} \bmod 97 &\equiv 61 \bmod 97 \end{aligned}$$

und damit:

$$7^{96} \bmod 97 \equiv 61 \cdot 35 \bmod 97 \equiv 1 \bmod 97$$

Rabin-Miller Test

Die bei weitem wichtigste Methode in den Implementierungen von kryptographischen Algorithmen ist der RABIN-MILLER Test. Dieser Algorithmus liefert eine sehr effektive und schnelle Methode um die Primalität einer Zahl mit kontrollierbar kleiner Fehlerwahrscheinlichkeit festzustellen. Der RABIN-MILLER Test wird auch *Pseudoprim Test* genannt. Dies ist ein Verfahren, welches für eine gegebene Zahl n folgenden Output liefert: Entweder *könnte die Zahl n eine Primzahl sein* oder *die Zahl n ist definitiv keine Primzahl.*

Der Algorithmus von MILLER-RABIN funktioniert folgendermaßen:

Wähle eine zufällige Zahl p; das Ziel ist zu prüfen, ob diese Zahl prim ist oder nicht. Dann berechnet man b, wobei b die Anzahl ist, wie oft 2 die Zahl $p-1$ teilt, i.e. 2^b ist die größte Potenz von 2, die $p-1$ teilt. Anschließend berechnet man die Zahl m so, dass

$$p = 1 + m \cdot 2^b$$

Anschließend führt man folgende Schritte aus:

1. Wähle eine zufällige Zahl a mit $a < p$
2. Setze $j = 0$ und $z = a^m \mod p$
3. Falls $z = 1$ oder $z = p-1$ dann besteht p den Test und könnte Primzahl sein.
4. Falls $j > 0$ und $z = 1$ dann ist p keine Primzahl.
5. Setze $j = j+1$. Falls $j < b$ und $z \neq p-1$ setze $z = z^2 \mod p$ und gehe zu Schritt 4. Falls $z = p-1$ dann besteht p den Test und könnte Primzahl sein.
6. Falls $j = b$ und $z \neq p-1$ dann ist p keine Primzahl.

Um zu verstehen, wie der Miller-Rabin Test funktioniert, müssen zunächst einige Resultate abgeleitet werden.

Behauptung:

Falls p eine ungerade Primzahl ist, dann hat die modulare Gleichung

$$x^2 \equiv 1 \mod p$$

die beiden Lösungen $x \equiv 1$ und $x \equiv -1$.

Beweis:

Gemäß Annahme gilt:

$$x^2 \equiv 1 \mod p,$$

daher ist

$$\begin{aligned} x^2 - 1 &\equiv 0 \mod p \\ (x-1) \cdot (x+1) &\equiv 0 \mod p \end{aligned}$$

Die letzte Kongruenz bedeutet, dass die Primzahl p Teiler von $x-1$ oder $x+1$ ist oder beide teilt. Angenommen, p dividiert beide Faktoren $x-1$ und $x+1$. Die impliziert, dass es zwei ganze Zahlen $k, l \in \mathbb{Z}$ gibt mit:

$$\begin{aligned} x + 1 &= k \cdot p \\ x - 1 &= l \cdot p \end{aligned}$$

Subtrahiert man diese beiden Gleichungen voneinander, folgt:

$$2 = (k - l) \cdot p$$

Diese Gleichung hat als Lösung nur $p = 2$, was ein Widerspruch zur Annahme ist, dass p eine ungerade Primzahl ist. Daher gilt für eine gegebene Lösung x: Entweder ist $p \mid (x + 1)$ oder $p \mid (x - 1)$ aber nicht beides.

Wir nehmen daher an, dass p Teiler von $x - 1$ ist, i.e. $p \mid (x - 1)$. Das bedeutet:

$$x - 1 = k \cdot p \qquad \text{für ein } k \in \mathbb{Z}$$

Daher ist $x \equiv 1 \bmod p$. Mit einer ähnlichen Argumentation findet man die andere Lösung.

Die obige Behauptung kann auch umgekehrt formuliert werden: Wenn es andere Lösungen der quadratischen modularen Gleichung $x^2 \equiv 1 \bmod n$ als ± 1 gibt, dann ist n keine Primzahl.

Beispiele:

1. Wir suchen die Lösungen der quadratischen modularen Gleichung

   $$x^2 \equiv 1 \bmod 7.$$

   Die einfachste Art, diese Gleichung zu lösen besteht darin, die Multiplikationstabelle von $\mathbb{Z}_7$ aufzustellen. Diese Tabelle ist in der Abbildung [19.1] dargestellt. Die Lösungen der obigen Gleichung findet man, in dem man in der i-ten Zeile zur i-ten Spalte geht. Wenn dort der Eintrag 1 steht, ist der EIntrag i eine Lösung der quadratischen Gleichung. Man liest folgendes ab:

   $$\begin{aligned} 1 \times 1 &= 1^2 \equiv 1 \bmod 7 \\ 2 \times 2 &= 2^2 \equiv 4 \bmod 7 \\ 3 \times 3 &= 3^2 \equiv 2 \bmod 7 \\ 4 \times 4 &= 4^2 \equiv 2 \bmod 7 \\ 5 \times 5 &= 5^2 \equiv 4 \bmod 7 \\ 6 \times 6 &= 6^2 \equiv 1 \bmod 7 \end{aligned}$$

   Die einzigen Kandidaten für x sind 1 und 6, da nur für diese Zahlen $x^2 \equiv 1 \bmod 7$ gilt. Da $6 \equiv -1 \bmod 7$, sind die einzigen beiden Lösungen $x = \pm 1$ (und 7 ist nach dem obigen Argumet eine Primzahl).

2. In einem zweiten Beispiel betrachten wir die quadratische modulare Gleichung in $\mathbb{Z}_8$, i.e. zu lösen ist:

   $$x^2 \equiv 1 \bmod 8$$

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Tabelle 19.1: Multiplikationstabelle von $\mathbb{Z}_7$.

Aus der Tabelle [19.2] erhalten wir durch Ablesen:

$$1^2 = 1 \equiv 1 \bmod 8$$
$$2^2 = 4 \equiv 4 \bmod 8$$
$$3^2 = 9 \equiv 1 \bmod 8$$
$$4^2 = 16 \equiv 0 \bmod 8$$
$$5^2 = 25 \equiv 1 \bmod 8$$
$$6^2 = 36 \equiv 4 \bmod 8$$
$$7^2 = 49 \equiv 1 \bmod 8$$

×	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Tabelle 19.2: Multiplikationstabelle von $\mathbb{Z}_8$.

Daraus erhalten wir die Lösungen $x = \pm 1, \pm 3$, da

$$1^2 \equiv 1 \bmod 8 \Longrightarrow 1 \equiv 1 \bmod 8$$
$$3^2 \equiv 1 \bmod 8 \Longrightarrow 3 \equiv 3 \bmod 8$$
$$5^2 \equiv 1 \bmod 8 \Longrightarrow 5 \equiv -3 \bmod 8$$
$$7^2 \equiv 1 \bmod 8 \Longrightarrow 7 \equiv -1 \bmod 8$$

Man erkennt also, dass in $\mathbb{Z}_8$ Lösungen x der modularen quadratischen Gleichung

$$x^2 \equiv 1 \bmod 8$$

existieren mit $x \neq \pm 1$, daher ist 8 keine Primzahl.

Wir können nun eine technische Version des Algorithmus von MILLER und RABIN (siehe [33]) angeben. Dieser Algorithmus testet eine gegebene Zahl auf ihre Primzahleigenschaft[13].

Algorithmus Miller-Rabin Test

MILL-RAB(a, n)

MR0 Input: n, a, x, d, i wobei
- n Zahl, die auf ihre Primzahleigenschaft getestet wird
- a zufällige Zahl zwischen 0 und $n-1$
- x, d, i Variable

MR1 Initialisierung
- Setze $d \leftarrow 1$
- $b_k b_{k-1} \ldots b_1 b_0$ ist die Binärdarstellung von $n-1$.
- Beginne mit $i = k$

MR2 Wiederhole die folgenden Schritte $k+1$ mal:

MR2a Ersetze $x \leftarrow d$

MR2b Berechne und ersetze $d \leftarrow (d \times d) \bmod n$

MR2c **if** $d = 1$ und $x \neq 1$ und $x \neq n-1$
then return TRUE STOP
else go to step **MR2d**

MR2d **if** $b_i = 1$
then $d \leftarrow (d \times a) \bmod n$
else go to step **MR2e**

MR2e Setze den Zähler auf $i \leftarrow i - 1$
if $i \geq 0$
then weiter mit **MR2a**
else weiter mit **MR3**

MR3 **if** $d \neq 1$
then return TRUE STOP
else return FALSE STOP

Der Input des MILLER-RABIN Tests sind zwei Zahlen n und a. n ist die Zahl, die auf die Primzahleigenschaft untersucht wird. a ist eine zufällige, ganze Zahl kleiner als n. Diese Zahl nennt man *Zeuge*[14]. Das Ziel ist zu testen, ob n prim ist. Falls der MILLER-RABIN Test den Rückgabewert `TRUE` liefert, dann ist die Zahl n definitic *nicht* prim. Falls der Rückgabewert `FALSE` ist, kann n eine Primzahl sein.

[13] Dieser Algorithmus wurde entwickelt in: MICHAEL O. RABIN, *Probabilistic Algorithm for Primality Testing*, Journal of Number Theory, Vol. 12, 1980, pp. 128 - 138, und G. L. MILLER, *Riemann's Hypothesis and Tests for Primality*. Journal of Computer Systems Science, Vol. 13, No. 3, Dec. 1976, pp 300 – 317.

[14] Engl.: *witness*.

Beispiele:

Um zu sehen, wie der MILLER-RABIN Test arbeitet, sehen wir uns zwei Beispiele im Detail an.

1. Die Inputzahlen unseres ersten Beispiels sind $n = 25$ und $a = 17$. Offensichtlich ist n keine Primzahl. Die Binärdarstellung von $n-1$ ist die Bitfolge `11 000`.

   Step1: MR0: Input:
   $n = 25, a = 17, x, d, i$

   Step2: MR1: Initialisierung:
   Setze $d = 1$ und

   $$b_4 = b_3 = 1, \quad b_2 = b_1 = b_0 = 0,$$

   die Indexvariable i wird auf den Wert $i = 4$ gesetzt.

   Step3: MR2: Wiederhole die folgenden Schritte $k + 1 = 5$ mal.

   Step4: MR2a: Setze $x \longleftarrow d = 1$.

   Step5: MR2b: Berechne

   $$d \leftarrow d^2 \bmod n = 1^2 \bmod 25 = 1 \quad \Longrightarrow d = 1$$

   Step6: MR2c: Abfrage:
   $d = 1$? Ja $x \neq 1$? nein $\Longrightarrow$ weiter mit Schritt **MR2d**

   Step7: MR2d: Prüfe $b_4 = 1$? Ja $\Longrightarrow$ setze

   $$d \leftarrow d \cdot a \bmod n = 17 \bmod 25 = 17.$$

   Also: $d = 17$.

   Step8: MR2e: Indexvariable i zurücksetzen:

   $$i \longleftarrow i - 1 = 3.$$

   $\hookrightarrow$ weiter mit Schritt **MR2a**.

   Step9: MR2a: Variable x neu belegen:

   $$x \longleftarrow d = 17.$$

   Also $d = 17$.

   Step10: MR2b: Berechne

   $$d \longleftarrow d^2 \bmod n = 17^2 \bmod 25 = 14 \quad \Longrightarrow d = 14$$

   Step11: MR2c: Abfrage:
   $d = 1$? Nein weiter mit Schritt **MR2d**.

   Step12: MR2d: Prüfe $b_3 = 1$? Ja $\Longrightarrow$ setze

   $$d \longleftarrow d \cdot a \bmod n = (14 \cdot 17) \bmod 25 = 13.$$

   Also: $d = 13$.

Step13: MR2e: Indexvariable zurücksetzen:

$$i \longleftarrow i - 1 = 2.$$

$\hookrightarrow$ weiter mit Schritt **MR2a**.

Step14: MR2a: Variable x neu belegen:

$$x \longleftarrow d = 13.$$

Also $d = 13$.

Step15: MR2b: Berechne

$$d \longleftarrow d^2 \bmod n = 13^2 \bmod 25 = 19 \quad \Longrightarrow \; d = 19$$

Step16: MR2c: Abfrage:
$d = 1$? Nein weiter mit Schritt **MR2d**.

Step17: MR2d: Prüfe: $b_2 = 1$? Nein $\Longrightarrow$ weiter mit Schritt **MR2e**

Step18: MR2e: Indexvariable i zurücksetzen:

$$i \longleftarrow i - 1 = 1.$$

$i \geq 0$ weiter mit Schritt **MR2a**.

Step19: MR2a: Variable x neu belegen:

$$x \longleftarrow d = 19.$$

Also $d = 19$.

Step20: MR2b: Berechne

$$d \leftarrow d^2 \bmod n = 19^2 \bmod 25 = 11 \quad \Longrightarrow d = 11$$

Step21: MR2c: Abfrage:
$d = 1$? Nein weiter mit Schritt **MR2d**.

Step22: MR2d: Prüfe: $b_1 = 1$? Nein $\hookrightarrow$ weiter mit Schritt **MR2e**.

Step23: MR2e: Indexvariable i zurücksetzen:

$$i \longleftarrow i - 1 = 0.$$

$i \geq 0$ weiter mit Schritt **MR2a**.

Step24: MR2a: Variable x neu belegen:

$$x \longleftarrow d = 11.$$

Also $d = 11$.

Step25: MR2b: Berechne:

$$d \leftarrow d^2 \bmod n = 11^2 \bmod 25 = 21 \quad \Longrightarrow d = 21.$$

Step26: MR2c: Abfrage:
$d = 1$? Nein $\hookrightarrow$ weiter mit Schritt **MR2d**.

Step27: MR2d: Prüfe: $b_0 = 1$? Nein $\hookrightarrow$ weiter mit Schritt **MR2e**.

Step28: MR2e: Indexvariable i zurücksetzen:

$$i \longleftarrow i - 1 = -1.$$

$i < 0$ $\hookrightarrow$ weiter mit Schritt **MR3**.

Step29: MR3: Prüfe $d \neq 1$? Ja $\Longrightarrow$ Return TRUE STOP

Daher ist $n = 25$ keine Primzahl.

2. Für das zweite Beispiel setzen wir $n = 17$ und $a = 5$. In diesem Fall ist n sicherlich eine Primzahl. Die Binärdarstellung von $n - 1 = 16$ ist:

$$16_{10} = 10000_2 = b_4 b_3 b_2 b_1 b_0.$$

Step1: MR0: Input:
$n = 17, a = 5, x, d, i$

Step2: MR1: Initialisierung:
Setze $d = 1$ und

$$b_4 = 1, \quad b_3 = b_2 = b_1 = b_0 = 0,$$

die Indexvariable i wird auf den Wert $i = 4$ gesetzt.

Step3: MR2: Wiederhole die folgenden Schritte $k + 1 = 5$ mal.

Step4: MR2a: Setze $x \longleftarrow d = 1$.

Step5: MR2b: Berechne

$$d \leftarrow d^2 \bmod n = 1^2 \bmod 17 = 1 \quad \Longrightarrow d = 1$$

Step6: MR2c: Abfrage:
$d = 1$? Ja $x \neq 1$? Nein $\Longrightarrow$ weiter mit Schritt **MR2d**

Step7: MR2d: Prüfe $b_4 = 1$? Ja $\Longrightarrow$ setze

$$d \leftarrow d \cdot a \bmod n = 5 \bmod 17 = 5.$$

Also: $d = 5$.

Step8: MR2e: Indexvariable i zurücksetzen:

$$i \longleftarrow i - 1 = 3.$$

$\hookrightarrow$ weiter mit Schritt **MR2a**.

Step9: MR2a: Variable x neu belegen:

$$x \longleftarrow d = 5.$$

Also $d = 5$.

Step10: MR2b: Berechne

$$d \longleftarrow d^2 \bmod n = 5^2 \bmod 17 = 8 \quad \Longrightarrow \; d = 8$$

Step11: MR2c: Abfrage:
$d = 1$? Nein weiter mit Schritt **MR2d**.

Step12: MR2d: Prüfe $b_3 = 1$? Nein, weiter mit Schritt **MR2e**.

Step13: MR2e: Indexvariable zurücksetzen:

$$i \longleftarrow i - 1 = 2.$$

$\hookrightarrow$ weiter mit Schritt **MR2a**.

Step14: MR2a: Variable x neu belegen:

$$x \longleftarrow d = 8.$$

Also $d = 13$.

Step15: MR2b: Berechne

$$d \longleftarrow d^2 \bmod n = 8^2 \bmod 17 = 13 \quad \Longrightarrow d = 13$$

Step16: MR2c: Abfrage:
$d = 1$? Nein weiter mit Schritt **MR2d**.

Step17: MR2d: Prüfe: $b_2 = 1$? Nein $\Longrightarrow$ weiter mit Schritt **MR2e**

Step18: MR2e: Indexvariable i zurücksetzen:

$$i \longleftarrow i - 1 = 1.$$

$i \geq 0$ weiter mit Schritt **MR2a**.

Step19: MR2a: Variable x neu belegen:

$$x \longleftarrow d = 13.$$

Also $d = 13$.

Step20: MR2b: Berechne

$$d \leftarrow d^2 \bmod n = 13^2 \bmod 17 = 16 \quad \Longrightarrow d = 16.$$

Step21: MR2c: Abfrage:
$d = 1$? Nein weiter mit Schritt **MR2d**.

Step22: MR2d: Prüfe: $b_1 = 1$? Nein $\hookrightarrow$ weiter mit Schritt **MR2e**.

Step23: MR2e: Indexvariable i zurücksetzen:

$$i \longleftarrow i - 1 = 0.$$

$i \geq 0$ weiter mit Schritt **MR2a**.

Step24: MR2a: Variable x neu belegen:

$$x \longleftarrow d = 16.$$

Also $d = 16$.

Step25: MR2b: Berechne:

$$d \leftarrow d^2 \bmod n = 16^2 \bmod 17 = 1 \quad \Longrightarrow d = 1.$$

Step26: MR2c: Abfrage:
$d = 1$? Ja $x \neq 1$? Ja $x \neq n - 1$? Nein $\hookrightarrow$ weiter mit Schritt **MR2d**.

Step27: MR2d: Prüfe: $b_0 = 1$? Nein $\hookrightarrow$ weiter mit Schritt **MR2e**.

Step28: MR2e: Indexvariable i zurücksetzen:

$$i \longleftarrow i - 1 = -1.$$

$i < 0$ $\hookrightarrow$ weiter mit Schritt **MR3**.

Step29: MR3: Prüfe $d \neq 1$? Nein $\Longrightarrow$ Return FALSE STOP

Daher ist die getestete Zahl $n = 17$ eine Primzahl.

Einige praktische Aspekte

Bei der praktischen Implementierung werden für die Erzeugung von Primzahlen folgende Regeln angewendet:

- In einem ersten Schritt wird eine zufällige n-Bit Zahl p erzeugt.
- In einem zweiten Schritt wird das höchstwertige Bit auf den Wert 1 gesetzt um zu garantieren, dass die Primzahl die gewünschte Länge hat[15]. Weiterhin wird das niedrigwertigste Bit auf den Wert 1 gesetzt, um zu garantieren, dass die Zahl ungerade ist[16].
- Der dritte Schritt besteht darin, zu prüfen, ob die so erzeugte Zahl durch eine kleine Primzahl teilbar ist. Typischerweise werden Teiler q bis 256 gewählt.
- Der vierte Schritt ist der RABIN-MILLER Test der Zahl p mit einer zufälligen Zahl a. Falls p den Test besteht, wählt man eine weitere zufällige Zahl a und testet erneut. Um die Rechenzeit zu reduzieren, sollte a eine kleine Zahl sein. Es wird empfohlen, den RABIN-MILLER Test fünf Mal zu durchlaufen. Falls p einen der Tests nicht besteht, wird eine neue zufällige Bitfolge generiert und das Prozedere beginnt von vorne.

Der dritte Schritt ist optional aber sehr zweckmäßig. Der Grund ist, wenn man eine ungerade, zufällige Zahl p testet, dass sie nicht durch 3, 5 und 7 teilbar ist, eliminiert man bereits 50% der ungeraden Zahlen, bevor man den RABIN-MILLER Test in Schritt 4 anwendet. Testet man p auf Teilbarkeit durch Primzahlen kleiner 100, dann werden etwa 75% der ungeraden Zahlen ausgeschlossen und bei Zahlen bis 256 sind 80 Prozent der Kandidaten eliminiert.

[15] Sinn und Zweck dieses Schritts erkennt man beispielsweise anhand der Zahl 3. Diese Zahl ist sicherlich eine Primzahl mit der Binärcodierung

$$\underbrace{00\ldots0}_{511}1$$

wenn man eine 512 Bit Primzahl generieren möchte. Dies ist jedoch für kryptographische Zwecke keine geeignete Primzahl.

[16] Alle Primzahlen bis auf 2 sind ungerade.

Kapitel 20

Diskreter Logarithmus

20.1 Primitive Wurzeln

Sei G eine multiplikative Gruppe mit Einselement 1_G und $a \in G$. Eine Zahl $m \in \mathbb{N}$ mit $a^m = 1$ und $a^r \neq 1$ für jedes $r \in \mathbb{N}, 0 < r < m$ nennt man **Ordnung** des Elements a in der Gruppe G. Wir betrachten in diesem Kapitel insbesondere die Gruppe $\mathbb{Z}_p^*$, wobei p eine Primzahl ist. Dann hat ein Element $a \in \mathbb{Z}_p^*$ die Ordnung m in $\mathbb{Z}_p^*$, wenn

$$\begin{aligned} a^m \bmod p &\equiv 1 \bmod p \\ a^r \bmod p &\not\equiv 1 \bmod p \quad \forall\, r \in \mathbb{N}, 0 < r < m \end{aligned}$$

Die Existenz der Ordnung m eines jeden Elements $a \in \mathbb{Z}_p^*$ ist durch das Theorem von FERMAT gesichert. Dieser Satz liefert

$$a^{p-1} \bmod p \equiv 1 \bmod p$$

und damit $m < p$.

Wir definieren folgende Menge:

$$\mathbb{Z}_p(a) = \{a^x \bmod p \mid 0 \leq x < m\}.$$

mit

$$a^0 \bmod p \equiv 1 \bmod p$$

Es gilt folgender Satz:

Theorem:

Sei p Primzahl und $a \in \mathbb{Z}_p^*$ mit Ordnung m. Dann gilt:

1. $\mid \mathbb{Z}_p(a) \mid = m$.
2. $\mathbb{Z}_p(a)$ ist zyklische Untergruppe von $\mathbb{Z}_p^*$
3. m ist Teiler von $p - 1$.

Beweis:

Zu 1: Wir nehmen an, es gibt zwei Zahlen, $k, l \in \mathbb{N}$ mit

$$0 \leq k < l < m$$

mit

$$a^k \bmod p \equiv a^l \bmod p$$

Dies ist äquivalent zu

$$a^{k-l} \bmod p \equiv 1 \bmod p$$

Wegen

$$0 < l - k < m$$

ist dies aber ein Widerspruch zu der Annahme, dass das Element a die Ordnung m in der Gruppe $\mathbb{Z}_p^*$ hat.

Zu 2: Wir zeigen zunächst, dass $\mathbb{Z}_p(a)$ eine multiplikative Gruppe ist. Die Abgeschlossenheit von $\mathbb{Z}_p(a)$ unter Multiplikation zeigt man folgendermaßen. Betrachte zwei beliebige Elemente

$$a^x \bmod p, a^y \bmod p \in \mathbb{Z}_p(a).$$

Dabei können wir annehmen, dass $x, y \in \mathbb{N}$ mit $0 \leq x, y, < m$. Das Produkt dieser beiden Elemente aus $\mathbb{Z}_p(a)$ ist unter Anwendung der modularen Arithmetik:

$$a^x \bmod p \cdot a^y \bmod p \equiv a^{x+y} \bmod p$$

Nun können zwei Fälle eintreten:

1. Falls $x + y < m$, dann ist $a^{x+y} \bmod p \in \mathbb{Z}_p(a)$. Dann ist $\mathbb{Z}_p(a)$ abgeschlossen unter der Multiplikation.
2. Falls $x + y \geq m$ ist, dann setzen wir:

   $$r = x + y - m < m$$

   Dann gilt:

   $$\begin{aligned} a^{x+y} \bmod p &\equiv a^{m+r} \bmod p \\ &\equiv (a^m \cdot a^r) \bmod p \\ &\equiv \Big[\big(a^m \bmod p\big) \cdot \big(a^r \bmod p\big) \Big] \bmod p \\ &\equiv a^r \bmod p, r < m \end{aligned}$$

   denn per Konstruktion gilt $a^m \bmod p \equiv 1 \bmod p$, i.e.

   $$a^{x+y} \bmod p \in \mathbb{Z}_p(a)$$

   Daher ist auch in diesem Fall $\mathbb{Z}_p(a)$ abgeschlossen unter der Multiplikation.

Wir müssen noch zeigen, dass zu jedem $b \in \mathbb{Z}_p(a)$ ein Element $b^{-1} \in \mathbb{Z}_p(a)$ existiert, mit

$$b \cdot b^{-1} = 1$$

Sei nun $b \in \mathbb{Z}_p(a)$ ein beliebiges Element. Dann existiert eine eindeutige Zahl $x \in \{0, 1, \ldots, m-1\}$ mit

$$b = a^x \bmod p$$

Dann ist das inverse Element gegeben durch:

$$c = a^{m-x} \bmod p$$

denn

$$\begin{aligned} b \cdot c &= (a^x \bmod p) \cdot (a^{m-x} \bmod p) \\ &\equiv a^{x+m-x} \bmod p \\ &\equiv a^m \bmod p \\ &\equiv 1 \bmod p \end{aligned}$$

Dies zeigt, dass $\mathbb{Z}_p(a)$ die von a erzeugte zyklische Untergruppe von $\mathbb{Z}_p^*$ ist.

Zu 3: Wir müssen zeigen, dass m ein Teiler von $p-1$ ist. Wir nehmen dazu an, es gibt geeignete Zahlen $k, r \in \mathbb{N}$ mit

$$p - 1 = k \cdot m + r, 0 \leq r < m.$$

m ist genau dann Teiler von $p-1$, wenn wir zeigen, dass $r = 0$ ist. Es gilt nach dem FERMATschen Satz:

$$\begin{aligned} 1 &= a^{p-1} \bmod p \\ &= a^{km+r} \bmod p \\ &\equiv (a^m)^k \cdot a^r \bmod p \\ &\equiv a^r \bmod p \\ &\Longleftrightarrow r = 0 \end{aligned}$$

Damit ist $p - 1 = m \cdot k$, also ist m Teiler von $p - 1$.

Beispiel:

Um mit den Begriffen und Konzepten dieses Theorems vertraut zu werden, betrachten wir ein konkretes Beispiel. Wir setzen $p = 13$ und untersuchen die Gruppe

$$G = (\mathbb{Z}_{13}^*, \cdot)$$

explizit. Wir betrachten zunächst das Element $2 \in \mathbb{Z}_{13}^*$ und bilden die Potenzen modulo 13:

$$
\begin{aligned}
2^0 \bmod 13 &\equiv 1 \bmod 13 \\
2^1 \bmod 13 &\equiv 2 \bmod 13 \\
2^2 \bmod 13 &\equiv 4 \bmod 13 \\
2^3 \bmod 13 &\equiv 8 \bmod 13 \\
2^4 \bmod 13 &\equiv 3 \bmod 13 \\
2^5 \bmod 13 &\equiv 6 \bmod 13 \\
2^6 \bmod 13 &\equiv 12 \bmod 13 \\
2^7 \bmod 13 &\equiv 11 \bmod 13 \\
2^8 \bmod 13 &\equiv 9 \bmod 13 \\
2^9 \bmod 13 &\equiv 5 \bmod 13 \\
2^{10} \bmod 13 &\equiv 10 \bmod 13 \\
2^{11} \bmod 13 &\equiv 7 \bmod 13 \\
2^{12} \bmod 13 &\equiv 1 \bmod 13
\end{aligned}
$$

Das Element $3 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
3^0 \bmod 13 &\equiv 1 \bmod 13 \\
3^1 \bmod 13 &\equiv 3 \bmod 13 \\
3^2 \bmod 13 &\equiv 9 \bmod 13 \\
3^3 \bmod 13 &\equiv 1 \bmod 13 \\
3^4 \bmod 13 &\equiv 3 \bmod 13 \\
&\vdots
\end{aligned}
$$

Das Element $4 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
4^0 \bmod 13 &\equiv 1 \bmod 13 \\
4^1 \bmod 13 &\equiv 4 \bmod 13 \\
4^2 \bmod 13 &\equiv 3 \bmod 13 \\
4^3 \bmod 13 &\equiv 12 \bmod 13 \\
4^4 \bmod 13 &\equiv 9 \bmod 13 \\
4^5 \bmod 13 &\equiv 10 \bmod 13 \\
4^6 \bmod 13 &\equiv 1 \bmod 13 \\
4^7 \bmod 13 &\equiv 4 \bmod 13 \\
&\vdots
\end{aligned}
$$

Das Element $5 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
5^0 \bmod 13 &\equiv 1 \bmod 13 \\
5^1 \bmod 13 &\equiv 5 \bmod 13 \\
5^2 \bmod 13 &\equiv 12 \bmod 13 \\
5^3 \bmod 13 &\equiv 8 \bmod 13 \\
5^4 \bmod 13 &\equiv 1 \bmod 13 \\
5^5 \bmod 13 &\equiv 5 \bmod 13 \\
5^6 \bmod 13 &\equiv 12 \bmod 13 \\
&\vdots
\end{aligned}
$$

Das Element $6 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
6^0 \bmod 13 &\equiv 1 \bmod 13 \\
6^1 \bmod 13 &\equiv 6 \bmod 13 \\
6^2 \bmod 13 &\equiv 10 \bmod 13 \\
6^3 \bmod 13 &\equiv 8 \bmod 13 \\
6^4 \bmod 13 &\equiv 9 \bmod 13 \\
6^5 \bmod 13 &\equiv 2 \bmod 13 \\
6^6 \bmod 13 &\equiv 12 \bmod 13 \\
6^7 \bmod 13 &\equiv 7 \bmod 13 \\
6^8 \bmod 13 &\equiv 3 \bmod 13 \\
6^9 \bmod 13 &\equiv 5 \bmod 13 \\
6^{10} \bmod 13 &\equiv 4 \bmod 13 \\
6^{11} \bmod 13 &\equiv 11 \bmod 13 \\
6^{12} \bmod 13 &\equiv 1 \bmod 13
\end{aligned}
$$

Das Element $7 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
7^0 \bmod 13 &\equiv 1 \bmod 13 \\
7^1 \bmod 13 &\equiv 7 \bmod 13 \\
7^2 \bmod 13 &\equiv 10 \bmod 13 \\
7^3 \bmod 13 &\equiv 5 \bmod 13 \\
7^4 \bmod 13 &\equiv 9 \bmod 13 \\
7^5 \bmod 13 &\equiv 11 \bmod 13 \\
7^6 \bmod 13 &\equiv 12 \bmod 13 \\
7^7 \bmod 13 &\equiv 6 \bmod 13 \\
7^8 \bmod 13 &\equiv 3 \bmod 13 \\
7^9 \bmod 13 &\equiv 8 \bmod 13 \\
7^{10} \bmod 13 &\equiv 4 \bmod 13 \\
7^{11} \bmod 13 &\equiv 2 \bmod 13 \\
7^{12} \bmod 13 &\equiv 1 \bmod 13
\end{aligned}
$$

Das Element $8 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
8^0 \bmod 13 &\equiv 1 \bmod 13 \\
8^1 \bmod 13 &\equiv 8 \bmod 13 \\
8^2 \bmod 13 &\equiv 12 \bmod 13 \\
8^3 \bmod 13 &\equiv 5 \bmod 13 \\
8^4 \bmod 13 &\equiv 1 \bmod 13 \\
8^5 \bmod 13 &\equiv 8 \bmod 13 \\
&\vdots
\end{aligned}
$$

Das Element $9 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
9^0 \bmod 13 &\equiv 1 \bmod 13 \\
9^1 \bmod 13 &\equiv 9 \bmod 13 \\
9^2 \bmod 13 &\equiv 3 \bmod 13 \\
9^3 \bmod 13 &\equiv 1 \bmod 13 \\
9^4 \bmod 13 &\equiv 9 \bmod 13 \\
&\vdots
\end{aligned}
$$

Das Element $10 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
10^0 \bmod 13 &\equiv 1 \bmod 13 \\
10^1 \bmod 13 &\equiv 10 \bmod 13 \\
10^2 \bmod 13 &\equiv 9 \bmod 13 \\
10^3 \bmod 13 &\equiv 12 \bmod 13 \\
10^4 \bmod 13 &\equiv 3 \bmod 13 \\
10^5 \bmod 13 &\equiv 4 \bmod 13 \\
10^6 \bmod 13 &\equiv 1 \bmod 13 \\
&\vdots
\end{aligned}
$$

Das Element $11 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
11^0 \bmod 13 &\equiv 1 \bmod 13 \\
11^1 \bmod 13 &\equiv 11 \bmod 13 \\
11^2 \bmod 13 &\equiv 4 \bmod 13 \\
11^3 \bmod 13 &\equiv 5 \bmod 13 \\
11^4 \bmod 13 &\equiv 3 \bmod 13 \\
11^5 \bmod 13 &\equiv 7 \bmod 13 \\
11^6 \bmod 13 &\equiv 12 \bmod 13 \\
11^7 \bmod 13 &\equiv 2 \bmod 13 \\
11^8 \bmod 13 &\equiv 9 \bmod 13 \\
11^9 \bmod 13 &\equiv 8 \bmod 13 \\
11^{10} \bmod 13 &\equiv 10 \bmod 13 \\
11^{11} \bmod 13 &\equiv 6 \bmod 13 \\
11^{12} \bmod 13 &\equiv 1 \bmod 13 \\
&\vdots
\end{aligned}
$$

Das Element $12 \in \mathbb{Z}_{13}^*$ liefert die Potenzen:

$$
\begin{aligned}
12^0 \bmod 13 &\equiv 1 \bmod 13 \\
12^1 \bmod 13 &\equiv 12 \bmod 13 \\
12^2 \bmod 13 &\equiv 1 \bmod 13 \\
&\vdots
\end{aligned}
$$

Zusammengefaßt erhalten wir also:

$$\begin{aligned}
\mathbb{Z}_{13}(2) &= \{2^x \bmod 13, 0 \leq x < 12 = m\} \\
&= \{1,2,3,4,5,6,7,8,9,10,11,12\} \\
\mathbb{Z}_{13}(3) &= \{3^x \bmod 13, 0 \leq x < 3 = m\} \\
&= \{1,3,9\} \\
\mathbb{Z}_{13}(4) &= \{4^x \bmod 13, 0 \leq x < 6 = m\} \\
&= \{1,3,4,9,12\} \\
\mathbb{Z}_{13}(5) &= \{5^x \bmod 13, 0 \leq x < 4 = m\} \\
&= \{1,5,8,12\} \\
\mathbb{Z}_{13}(6) &= \{6^x \bmod 13, 0 \leq x < 12 = m\} \\
&= \{1,2,3,4,5,6,7,8,9,10,11,12\} \\
\mathbb{Z}_{13}(7) &= \{7^x \bmod 13, 0 \leq x < 12 = m\} \\
&= \{1,2,3,4,5,6,7,8,9,10,11,12\} \\
\mathbb{Z}_{13}(8) &= \{8^x \bmod 13, 0 \leq x < 4 = m\} \\
&= \{1,5,8,12\} \\
\mathbb{Z}_{13}(9) &= \{9^x \bmod 13, 0 \leq x < 4 = m\} \\
&= \{1,3,9\} \\
\mathbb{Z}_{13}(10) &= \{10^x \bmod 13, 0 \leq x < 6 = m\} \\
&= \{1,3,4,9,10,12\} \\
\mathbb{Z}_{13}(11) &= \{11^x \bmod 13, 0 \leq x < 12 = m\} \\
&= \{1,2,3,4,5,6,7,8,9,10,11,12\} \\
\mathbb{Z}_{13}(12) &= \{12^x \bmod 13, 0 \leq x < 2 = m\} \\
&= \{1,12\}
\end{aligned}$$

Aus diesen expliziten Mengen erkennt man

- Die Zahlen $2, 6, 7, 11$ haben die Ordnung 12 in der Gruppe $(\mathbb{Z}_{13}^*, \cdot)$.
- Die Zahl 12 hat die Ordnung 2 in der Gruppe $(\mathbb{Z}_{13}^*, \cdot)$.
- Die Zahlen $3, 9$ haben die Ordnung 3 in der Gruppe $(\mathbb{Z}_{13}^*, \cdot)$.
- Die Zahlen $5, 8$ haben die Ordnung 4 in der Gruppe $(\mathbb{Z}_{13}^*, \cdot)$.
- Die Zahlen $4, 10$ haben die Ordnung 6 in der Gruppe $(\mathbb{Z}_{13}^*, \cdot)$.

Weiterhin liest man die Mächtigkeiten der Mengen $\mathbb{Z}_{13}(a)$ ab:

$$\begin{aligned}
|\,\mathbb{Z}_{13}(2)\,| &= |\,\mathbb{Z}_{13}(6)\,| = |\,\mathbb{Z}_{13}(7)\,| = |\,\mathbb{Z}_{13}(11)\,| = 12 \\
|\,\mathbb{Z}_{13}(12)\,| &= 2 \\
|\,\mathbb{Z}_{13}(3)\,| &= |\,\mathbb{Z}_{13}(9)\,| = 3 \\
|\,\mathbb{Z}_{13}(5)\,| &= |\,\mathbb{Z}_{13}(8)\,| = 4 \\
|\,\mathbb{Z}_{13}(4)\,| &= |\,\mathbb{Z}_{13}(8)\,| = |\,\mathbb{Z}_{13}(10)\,| = 6
\end{aligned}$$

Wie im obigen Satz bewiesen, korrespondieren die Mächtigkeiten der Mengen mit dem jeweiligen m. Alle Ordnungen $m = 2, 3, 4, 6, 12$ sind Teiler von $p - 1 = 12$.

Aus dieser Auflistung erkennt man weiterhin, dass die vier Zahlen $2, 6, 7$ und 11 die komplette Gruppe $(\mathbb{Z}_{13}^*, \cdot)$ erzeugen.

Alle Mengen $\mathbb{Z}_{13}(a)$ bilden zyklische Untergruppen von $\mathbb{Z}_{13}^*$. Wir zeigen dies hier exemplarisch anhand der Menge $\mathbb{Z}_{13}(4)$.

1. Die Menge

$$\mathbb{Z}_{13}(4) = \{1, 3, 4, 9, 10, 12\}$$

   ist abgeschlossen unter der Multiplikation. Dies zeigt man am einfachsten durch die Verknüpfungstafel.

$\cdot$	**1**	**3**	**4**	**9**	**10**	**12**
1	1	3	4	9	10	12
3	3	9	12	1	4	10
4	4	12	3	10	1	9
9	9	1	10	3	12	4
10	10	4	1	12	9	3
12	12	10	9	4	3	1

2. Das Einselement 1 der Gruppe $(\mathbb{Z}_{13}^*, \cdot)$ ist auch Element der Menge $\mathbb{Z}_{13}(4)$ und es gilt die Mengeninklusion

$$\mathbb{Z}_{13}(4) \subset \mathbb{Z}_{13}^*$$

3. Aus obiger Verknüpfung erhalten wir:

$$\begin{aligned} 3^{-1} &\equiv 9 \bmod 13 \\ 4^{-1} &\equiv 10 \bmod 13 \\ 9^{-1} &\equiv 3 \bmod 13 \\ 10^{-1} &\equiv 4 \bmod 13 \\ 12^{-1} &\equiv 12 \bmod 13 \end{aligned}$$

   i.e. $\forall\, a \in \mathbb{Z}_{13}(4)$ folgt $a^{-1} \in \mathbb{Z}_{13}(4)$.

4. Das Element 4 ist ein erzeugendes Element der Gruppe $(\mathbb{Z}_{13}(4), \cdot)$.

Damit ist gezeigt, dass $\mathbb{Z}_{13}(4)$ zyklische Untergruppe von $\mathbb{Z}_{13}^*$ ist.

Anmerkung:

Das zu

$$b = a^x \bmod p \ \in \mathbb{Z}_p(a)$$

inverse Element ist nach dem Beweis des obigen Theorems durch

$$b^{-1} = a^{m-x} \bmod p$$

gegeben. Im Beispiel haben wir die Inversen einfach durch Ablesen aus der Tabelle erhalten. Alternativ bestimmt man die Inverse mit Hilfe dieses Ausdrucks über:

$$\begin{aligned}
3 = 4^2 \bmod 13 &\Longrightarrow 3^{-1} = 4^{6-2} \bmod 13 = 9 \bmod 13 \\
4 = 4^1 \bmod 13 &\Longrightarrow 4^{-1} = 4^{6-1} \bmod 13 = 10 \bmod 13 \\
9 = 4^4 \bmod 13 &\Longrightarrow 9^{-1} = 4^{6-4} \bmod 13 = 3 \bmod 13 \\
10 = 4^5 \bmod 13 &\Longrightarrow 10^{-1} = 4^{6-5} \bmod 13 = 4 \bmod 13 \\
12 = 4^3 \bmod 13 &\Longrightarrow 12^{-1} = 4^{6-3} \bmod 13 = 12 \bmod 13
\end{aligned}$$

Definition:

Sei p eine Primzahl. Ein Element $g \in \mathbb{Z}_p^*$ habe die Ordnung $p-1$. Dann heißt g *primitive Wurzel modulo p*.

Beispiel:

Wie die explizite Berechnung im obigen Beispiel zeigt, hat die Gruppe $\mathbb{Z}_{13}^*$ genau vier verschiedene primitive Wurzeln, i.e. Elemente mit Ordnung $m = p-1$. Diese sind

$$2, 4, 6, 7$$

Wie das Beispiel ebenfalls zeigt, ist die von den primitiven Wurzeln erzeugte zyklische Untergruppe $(\mathbb{Z}_{13}(g), \cdot), g \in \{2, 4, 6, 7\}$ identisch mit der Gruppe $(\mathbb{Z}_{13}^*, \cdot)$. Dieses Ergebnis formulieren wir in folgendem Satz.

Theorem:

Sei p Primzahl und g eine primitive Wurzel modulo p. Dann gilt:

$$\mathbb{Z}_p(g) = \mathbb{Z}_p^*$$

Beweis:

Nach dem obigen Satz gilt:

$$| \mathbb{Z}_p(a) | = p - 1 = | \mathbb{Z}_p^* |$$

i.e. die Mengen $\mathbb{Z}_p(g)$ und $\mathbb{Z}_p^*$ sind gleich mächtig. Da nach obigen Satz $\mathbb{Z}_p(g)$ eine Untergruppe von $\mathbb{Z}_p^*$ ist, folgt:

$$\mathbb{Z}_p(g) = \mathbb{Z}_p^*$$

Mit anderen Worten, eine primitive Wurzel g modulo p generiert die komplette Gruppe $\mathbb{Z}_p^*$. Aus der Existenz einer primitiven Wurzel folgt aus den obigen beiden Sätzen, dass die Gruppe $\mathbb{Z}_p^*$ zyklisch ist. Mit anderen Worten,

$$(\mathbb{Z}_p^*, \cdot) = (< g >, \cdot)$$

Ist umgekehrt die Gruppe $(\mathbb{Z}_p^*, \cdot)$ zyklisch, dann gibt es ein Element g mit

$$\mathbb{Z}_p^* = \left\{ g^1 \bmod p, g^2 \bmod p, \ldots, g^{p-1} \bmod p \right\}$$

Dies zeigt, dass die Existenz einer primitiven Wurzel modulo p äquivalent dazu ist, dass die Gruppe $(\mathbb{Z}_p^*, \cdot)$ zyklisch ist.

Das folgende Theorem liefert eine Aussage darüber, wieviele verschiedene primitive Wurzeln eine multiplikative Gruppe hat. Weiterhin wird ein Kriterium bewiesen, wann ein Element einer Gruppe eine primitive Wurzel ist.

Theorem:

Sei $p > 2$ eine Primzahl. Dann gilt:

1. Die Anzahl der primitiven Wurzeln modulo p ist $\phi(p-1)$, wobei $\phi(p-1)$ die EULERsche Totientenfunktion ist.
2. Ein Element $a \in \mathbb{N}$ ist genau dann eine primitive Wurzel modulo p, wenn

$$a^{\frac{p-1}{q}} \bmod p \not\equiv 1 \bmod p$$

für jeden Primfaktor q von $p-1$ gilt. Falls die Anzahl k der verschiedenen Primfaktoren von $p-1$ bekannt ist, kann der Test, ob ein gegebenes $a \in \mathbb{N}$, $a \leq p-1$, eine primitive Wurzel modulo p ist in

$$O(k \cdot \log p) = O(\log^2 p)$$

Schritten durchgeführt werden.

Beweis:

Zu 1: Nach den Überlegungen, die dem Theorem vorangehen, existiert eine primitive Wurzel g modulo p. Für die primitive Wurzel gilt:

$$g^x \bmod p \equiv 1 \bmod p$$

genau für diese x, die ein Vielfaches von $p-1$ sind, i.e. für alle $x \in \mathbb{N}$ mit

$$x = (p-1) \cdot y, \quad y \in \mathbb{N}$$

Für jedes $n \in \{0, 1, 2, \ldots, p-2\}$ ist

$$b = g^n \bmod p$$

genau dann keine primitive Wurzel modulo p, wenn ein $m \in \{1, 2, \ldots, p-2\}$ existiert, mit

$$b^m \bmod p \equiv g^{mn} \bmod p \equiv 1 \bmod p$$

Da g eine primitive Wurzel ist, ist die nach dem zu Beginn des Beweises angegebenen Eigenschaft äquvalent zu

$$(p-1) \mid m \cdot n$$

Wegen $m < p-1$ muss ein Primfaktor von $p-1$ auch in n vorkommen. Aus der Eigenschaft, dass $(p-1)$ Teiler von $n \cdot m$ ist, folgt somit:

$$\mathrm{ggT}(p-1, n) > 1$$

Gilt umgekehrt

$$t = \mathrm{ggT}(p-1, n) > 1$$

so gelten für $m = (p-1)/t$ die Eigenschaften

- ✔ $1 \leq m < p-1$
- ✔ $n \cdot m = \frac{n}{t} \cdot (p-1)$

i.e. $(p-1) \mid n \cdot m$.

Negiert man diese Resultate, erhält man die Aussage: Für jedes $n \in \{0, 1, 2, \ldots, p-2\}$ ist

$$b = g^n \bmod p$$

genau dann eine primitive Wurzel modulo p, wenn

$$\text{ggT}(p-1, n) = 1$$

Es gibt aber genau $\phi(p-1)$ Zahlen mit dieser Eigenschaft, was die Aussage beweist.

Zu 2: Wenn a die Ordnung m in $\mathbb{Z}_p^*$ hat, dann ist m Teiler von $p-1$. Dabei ist a genau dann keine primitive Wurzel modulo p, wenn $m < p-1$ gilt. Dies ist jedoch genau dann der Fall, wenn

$$m \mid \frac{p-1}{q}$$

für einen Primfaktor q von $p-1$ gilt, also genau für

$$a^{\frac{p-1}{q}} \bmod p \equiv 1 \bmod p$$

Somit ist a genau dann eine primitive Wurzel, wenn

$$a^{\frac{p-1}{q}} \bmod p \not\equiv 1 \bmod p$$

für jeden Primfaktor q von $p-1$ gilt.

Beispiel:

Das oben explizite Beispiel betrifft die Gruppe

$$(\mathbb{Z}_p^*, \cdot) = (\mathbb{Z}_{13}^*, \cdot)$$

Damit ist $p-1 = 12$. Gemäß Abschnitt [14.5] ist die EULERsche Totientenfunktion

$$\phi(12) = 4$$

denn

$$\text{ggT}(1, 12) = \text{ggT}(5, 12) = \text{ggT}(7, 12) = \text{ggT}(11, 12) = 1$$

Wie wir bereits aus der expliziten Berechnung der Potenzen modulo 13 wissen, hat die Gruppe $(\mathbb{Z}_{13}^*, \cdot)$ vier primitive Wurzeln.

Achtung: Die zu $p-1 = 12$ coprimen Zahlen 1,5,7,11 sind nicht mit den primitiven Wurzeln gleichzusetzen!

Da $12 = 2^2 \cdot 3$ hat $p-1 = 12$ die beiden Primfaktoren $q_1 = 2$ und $q_2 = 3$. Damit prüft man beispielsweise:

$$2^{\frac{p-1}{q_1}} \bmod 13 = 2^6 \bmod 13 \equiv 12 \bmod 13 \not\equiv 1 \bmod 13$$
$$2^{\frac{p-1}{q_2}} \bmod 13 = 2^4 \bmod 13 \equiv 3 \bmod 13 \not\equiv 1 \bmod 13$$

woraus nach obigen Theorem folgt, dass 2 eine primitive Wurzel modulo 13 ist. Andererseits ist

$$3^{\frac{p-1}{q_1}} \bmod 13 = 3^6 \bmod 13 \equiv 1 \bmod 13$$
$$3^{\frac{p-1}{q_2}} \bmod 13 = 3^4 \bmod 13 \equiv 3 \bmod 13$$

daher kann 3 keine primitive Wurzel sein.

20.2 Berechnung diskreter Logarithmen

Wir nehmen an, p ist eine Primzahl und g ist eine primitive Wurzel modulo p. Der diskrete Logarithmus kann analog zum Logarithmusbegriff über den reellen Zahlen folgendermaßen definiert werden.

> Es sei ein Element $b \in \mathbb{Z}_p^*$ gegeben. Finde den eindeutigen Exponenten x, $x \in \{0, 1, 2, \ldots, p-1\}$ mit
>
> $$b = g^x \bmod p.$$

Gelegentlich bezeichnet man den diskreten Logarithmus auch als **Index**, i.e.

$$x = \text{ ind }_{g,p}(b) = \log_g(b)$$

Das **Diskrete Logarithmus Problem** besteht darin, diesen eindeutigen Exponenten x zu finden.

Einige Eigenschaften des Logarithmus über den reellen Zahlen zu einer beliebigen Basis $x \in \mathbb{R}$ sind:

$$\begin{aligned} \log_x(1) &= 0 \\ \log_x(x) &= 1 \\ \log_x(a \cdot b) &= \log_x(a) + \log_x(b) \\ \log_x(a^b) &= b \cdot \log_x(a) \\ &\text{für } a, b \in \mathbb{R}^+ \end{aligned}$$

Wir zeigen nun, dass der diskrete Logarithmus analoge Eigenschaften auf der Menge $\mathbb{Z}_p^*$ hat.

- Es gilt:

  $$\text{ind}_{g,p}(1) = 0$$

  da

  $$1 = g^0 \bmod p$$

- Es gilt:

  $$\text{ind}_{g,p}(g) = 1$$

  denn

  $$g = g^1 \bmod p$$

- Zu zeigen ist:

  $$\text{ind}_{g,p}(a \cdot b) = \text{ind}_{g,p}(a) + \text{ind}_{g,p}(b)$$

  für $a, b \in \mathbb{Z}_p^*$.

In den folgenden Abschnitten sehen wir uns eine Reihe von Algorithmen an, die dazu dienen, den diskreten Logarithmus zu berechnen.

20.2.1 'Brute-Force' Verfahren

Die einfachste Art und Weise, den diskreten Logarithmus zu berechnen besteht darin, alle Zahlen x von 0 bis $p-1$ zu prüfen, ob sie die Gleichung

$$b = g^x \bmod p.$$

erfüllen. Bei vollständiger Suche wird dieses Problem in der Zeit $O(p)$ gelöst.

20.2.2 Der Algorithmus von Shanks

Falls alle möglichen Paare

$$(x, g^x \bmod p)$$

vorab berechnet und dann nach der zweiten Komponente sortiert werden, kann das Problem in der Zeit $O(p)$ Vorabberechnungen und $O(p)$ Speicherplatz gelöst werden. Diese Komplexität ist unbefriedigend. Eine bessere Laufzeit hat der Algorithmus von SHANKS.

Baby-Step-Giant-Step Algorithmus von Shanks

Input: Eine primitve Wurzel modulo p, ein Element $b \in \mathbb{Z}_p^*$

Output: Der diskrete Logarithmus $x = \log_g b$.

① Berechne

$$m = \lceil \sqrt{p-1} \rceil$$

② Berechne $g^{m \cdot j} \bmod p$ für alle $j = 0, 1, \ldots, m-1$.

③ Sortiere die m geordneten Paare

$$(j, g^{j \cdot m} \bmod p)$$

nach ihrer zweiten Komponenten. Daraus resultiert eine Liste L_1.

④ Berechne

$$(b \cdot g^{-i}) \bmod p = (b \cdot g^{p-1-i}) \bmod p \quad \forall\, i = 0, 1, 2, \ldots, m-1$$

⑤ Sortiere die m geordneten Paare

$$(i, bg^{-i} \bmod p)$$

nach ihrer zweiten Komponenten. Daraus resultiert eine Liste L_2.

⑥ Finde ein Paar $(j, y) \in L_1$ und ein Paar $(i, y) \in L_2$.

⑦ Setze

$$\log_g b = (mj + i) \bmod p - 1$$

Theorem

Der Baby-Step-Giant-Step Algorithmus von SHANKS ist korrekt.

Beweis:

Wir müssen zeigen, dass der obige Algorithmus stets den diskreten Logarithmus liefert.

Gilt

$$(j, y) \in L_1 \text{ und } (i, y) \in L_2$$

dann folgt

$$y = g^{m \cdot j} \bmod p \stackrel{!}{=} (b \cdot g^{-i}) \bmod p$$

oder

$$g^{m \cdot j + i} \bmod p = b$$

Beispiel:

Um die Arbeitsweise des SHANKS Algorithmus zu verstehen, spielen wir das Verfahren mit kleinen Zahlen durch.

Wir wählen die Primzahl $p = 677$; der erste Schritt besteht darin, eine primitive Wurzel modulo p zu finden. Wir zeigen, dass $g = 8$ eine primitive Wurzel modulo 677 ist. Dies zeigen wir durch das Argument, dass g genau dann primitive Wurzel modulo p ist, wenn

$$g^{\frac{p-1}{q}} \bmod p \not\equiv 1 \bmod p$$

für jeden Primfaktor q von $p - 1$ gilt. Da

$$p - 1 = 676 = 2^2 \cdot 13^2$$

hat $p - 1 = 676$ die beiden Primfaktoren $q_1 = 2$ und $q_2 = 13$.

Dann folgt:

$$\begin{aligned} g^{\frac{p-1}{q_1}} \bmod p &= 8^{\frac{676}{2}} \bmod 677 \\ &= 8^{338} \bmod 677 \end{aligned}$$

Nun wendet man den `fastexp` Algorithmus an:

$$338 = 256 + 64 + 16 + 2$$

und

$$\begin{aligned}
8^2 &\bmod 677 \equiv 64 \bmod 677\\
8^4 &\bmod 677 \equiv 34 \bmod 677\\
8^8 &\bmod 677 \equiv 479 \bmod 677\\
8^{16} &\bmod 677 \equiv 615 \bmod 677\\
8^{32} &\bmod 677 \equiv 459 \bmod 677\\
8^{64} &\bmod 677 \equiv 134 \bmod 677\\
8^{128} &\bmod 677 \equiv 354 \bmod 677\\
8^{256} &\bmod 677 \equiv 71 \bmod 677\\
8^{512} &\bmod 677 \equiv 302 \bmod 677
\end{aligned}$$

Damit:

$$\begin{aligned}
g^{\frac{p-1}{q_1}} \bmod p &= 8^{\frac{676}{2}} \bmod 677\\
&= 8^{338} \bmod 677\\
&\equiv \left(71 \cdot 134 \cdot 615 \cdot 64\right) \bmod 677\\
&\equiv 676 \bmod 677\\
&\not\equiv 1 \bmod 677
\end{aligned}$$

Der zweite Primfaktor $q_2 = 13$ ergibt:

$$\begin{aligned}
g^{\frac{p-1}{q_2}} \bmod p &= 8^{\frac{676}{13}} \bmod 677\\
&= 8^{52} \bmod 677\\
&= 8^{32+16+4} \bmod 677\\
&\equiv (459 \cdot 615 \cdot 34) \bmod 677\\
&\equiv 538 \bmod 677\\
&\not\equiv 1 \bmod 677
\end{aligned}$$

Dies zeigt, dass $g = 8$ in der Tat eine primitive Wurzel modulo 677 ist.

Wir wollen nun mit Hilfe des Algorithmus von SHANKS den Wert $\log_8 555$ berechnen. Anders ausgedrückt, gesucht ist die Zahl x mit

$$555 \equiv 8^x \bmod 677$$

Als Input in den Algorithmus von SHANKS stehen also folgende Werte zur Verfügung:

$$g = 8; b = 555 \text{ und } m = \lceil\sqrt{676}\rceil = 26$$

Wir haben zuerst die Liste L_1 zu berechnen, i.e. alle Zahlenpaare der Form

$$(j, g^{m\cdot j} \bmod p) = (j, 8^{26\cdot j} \bmod 677) = (j, 344^j \bmod 677)$$

mit $j = 0, 1, \ldots, 25$.

$(0,1)$	$(1,344)$	$(2,538)$	$(3,251)$	$(4,365)$
$(5,315)$	$(6,40)$	$(7,220)$	$(8,533)$	$(9,562)$
$(10,383)$	$(11,414)$	$(12,246)$	$(13,676)$	$(14,333)$
$(15,139)$	$(16,426)$	$(17,312)$	$(18,362)$	$(19,637)$
$(20,457)$	$(21,144)$	$(22,115)$	$(23,294)$	$(24,263)$
$(25,431)$				

Nach Sortierung dieser Liste nach der zweiten Komponente resultiert in der Liste L_1.

Die zweite Liste enthält alle geordneten Paare der Form

$$(i, bg^{-i} \bmod p) = (i, 555 \cdot (8^i)^{-1} \bmod 677) = (i, 555 \cdot 8^{676-i} \bmod 677)$$

für die Werte $i = 0, 1, 2, \ldots, 25$.

$(0,555)$	$(1,154)$	$(2,527)$	$(3,489)$	$(4,315)$
$(5,124)$	$(6,354)$	$(7,552)$	$(8,69)$	$(9,601)$
$(10,329)$	$(11,295)$	$(12,460)$	$(13,396)$	$(14,388)$
$(15,387)$	$(16,133)$	$(17,609)$	$(18,330)$	$(19,549)$
$(20,661)$	$(21,675)$	$(22,169)$	$(23,275)$	$(24,119)$
$(25,438)$				

Nach Sortierung dieser Liste erhält man L_2.

Nun werden beide Listen durchlaufen, bis man zwei Paare $(j, y) \in L_1$ und $(i, y) \in L_2$ findet, die also die gleiche zweite Komponente haben. Dies ist das Paar $(5, 315) \in L_1$ und $(4, 315) \in L_2$. Gemäß dem letzten Schritt des SHANKS Algorithmus erhält man den diskreten Logarithmus zu:

$$\text{ind}_{8,677}(555) = (m \cdot j + i) \bmod 676 = (26 \cdot 5 + 4) \bmod 676 = 134.$$

Check:

Man verifiziert leicht, dass gilt:

$$\begin{aligned} 8^{134} \bmod 677 &\equiv 8^{128+4+2} \bmod 677 \\ &\equiv 354 \cdot 34 \cdot 64 \bmod 677 \\ &\equiv 555 \bmod 677. \end{aligned}$$

20.2.3 Der Pohlig-Hellman Algorithmus

Ein weiteres Verfahren zur Berechnung des diskreten Logarithmus stammt von S. POHLIG und MARTIN E. HELLMAN

Kapitel 21

Komplexität von Berechnungen

21.1 Die Länge einer Zahl

In diesem Kapitel[1] nehmen wir an, dass die Zahlen, mit denen wir arbeiten in Binärdarstellung vorliegen. Die komplette Arithmetik wird ebenfalls zur Basis 2 durchgeführt.

Unter der **Länge einer Zahl** n verstehen wir die Anzahl der Bits, die benötigt werden, die Zahl n darzustellen, i.e.

$$\text{Länge}(n) = 1 + \lfloor \log_2\ n \rfloor = 1 + \lfloor \frac{\ln n}{\ln 2} \rfloor,$$

dies kann mit dem Aufwand $O(\ln n)$ bestimmt werden.

Beispiele:

1. Sei $n = 47\,837$, dann benötigt man die folgende Anzahl von Bits

   $$\text{Länge}(47\,837) = 1 + \lfloor \log_2\ 47\,837 \rfloor = 1 + \lfloor 15.54 \rfloor = 16$$

   um die Zahl 47 837 im Binärsystem darzustellen.

2. Gegeben sei die Zahl $n = 1\,456\,789\,101$, hierfür sind

   $$\text{Länge}(1\,456\,789\,101) = 1 + \lfloor \log_2\ 1\,456\,789\,101 \rfloor = 1 + \lfloor 30.4401 \rfloor = 31$$

   Bits notwendig.

In diesem Abschnitt sehen wir uns an, wie man die Länge einer Zahl bestimmt, die das Resultat irgendwelcher arithmetischen Operationen ist. Dabei ist die Länge des Inputs bekannt.

Beispiel:

Wie groß ist die Länge einer Zahl, die erhalten wird als Resultat

[1] Diese Ausführungen folgen dem Buch von Neal Koblitz [130], Chapter 2.

a) der Addition und

b) der Multiplikation

von n positiven Zahlen, von denen jede höchstens die Länge k hat.

Um diese Aufgabe zu lösen, müssen wir darüber nachdenken, wie die Addition und die Multiplikation die Länge einer Zahl beeinflusst. Wenn man zwei Zahlen addiert, dann hat die Summe entweder die Länge der größeren Zahl oder die Länge 1 plus die Länge der größeren Zahl.

Addiert man n Zahlen, wobei jede dieser Zahlen höchstens die Länge k hat – i.e. jede Zahl ist kleiner 2^k – dann wird die Summe kleiner $n \cdot 2^k$ sein. Damit wird die Länge der Summe dieser n Zahlen höchstens $k +$ Länge (n) sein.

Beispiel:

Sei a die $k = 7$-Bit Zahl

$$a = \texttt{100 1101}$$

und b die $l = 7$-Bit Zahl

$$b = \texttt{100 0000}.$$

Dann ist die Summe dieser beiden Zahlen die 8-Bit Zahl:

$$a + b = \texttt{1000 1101},$$

daher ist Länge$(a + b) = 8$.

Um die Länge des Produkts von Zahlen zu handhaben, benutzen wir die Tatsache, dass für eine Zahl m der Länge k gilt:

$$2^{k-1} \leq m < 2^k.$$

Hat also die Zahl m_1 die Länge k und die Zahl m_2 die Länge l, dann können wir die beiden Ungleichungen

$$2^{k-1} \leq m_1 < 2^k$$
$$2^{l-1} \leq m_2 < 2^l$$

multiplizieren und erhalten

$$2^{k+l-2} \leq m_1 m_2 < 2^{k+l}.$$

Hieraus folgt, dass die Länge von $m_1 \cdot m_2$ entweder gleich der Summe der Längen von m_1 und m_2 ist, oder 1 weniger als die Summe der Längen von m_1 und m_2. Grob gesprochen, wenn man Zahlen multipliziert, addieren sich die Längen. Mit anderen Worten, die Längen verhalten sich bei der Multiplikation wie der Logarithmus.

Nun sollen n k-Bit Zahlen $m_1, m_2, \ldots, m_n$ multipliziert werden. Sind alle n Zahlen gleich, entspricht dies der Berechnung der n-ten Potenz der k-Bit Zahl m. Multiplizieren wir alle Ungleichungen

$$2^{k-1} \leq m_i < 2^k, \quad i = 1, 2, \ldots, n$$

dann erhalten wir

$$2^{nk-n} \leq \prod m_i < 2^{kn},$$

i.e. die Länge des Produkts ist zwischen $nk - (n-1)$ und nk.

Üblicherweise ist man nicht an der exakten Länge interessiert sondern lediglich an einer oberen Schranke. In diesem Fall kann man einfach sagen, dass die Multiplikation von n Zahlen mit maximaler Länge k auf eine Zahl führt mit einer Länge höchstens nk.

Beispiel:

Man finde die Länge von $n!$

Lösung:

Wir wollen hier eine einfache Bestimmung der Länge von $n!$ in der Form $O(g(n))$ ableiten. Da jede der n Zahlen, die zu $n!$ multipliziert werden, alle kleinere Längen als n haben, können wir das obige Argument anwenden und folgern:

$$\text{Länge } (n!) \leq n \cdot \text{Länge } (n) = O(n \ln n).$$

Beispiel:

Sei $n = 10$. Dann ist

$$\text{Länge } (n!) = O(10 \cdot \ln 10)$$

Check:

$$10! = 3\,628\,880$$

Um diese Zahl in binärer Darstellung zu erhalten – also in obiger Terminologie, die Länge von $n!$ – benötigt man

$$\log_2 3\,628\,880 \approx 21.791$$

Bits, dies ist $O(10 \ln 10)$.

Übungen:

1. Angenommen, eine k-Bit Zahl a wird durch eine l-Bit Zahl b dividiert (mit $l \leq k$), was auf einen Quotienten q und einen Rest r führt:

   $$a = qb + r, \qquad 0 \leq r < b.$$

   Was ist die Länge von q?

21.2 Laufzeitbestimmungen

Wie im vorherigen Abschnitt nehmen wird alle arithmetischen Operationen im Binärsystem vor.

21.2.1 Bit-Operationen

Wir beginnen mit der Untersuchung eines sehr einfachen arithmetischen Problems, der Addition zweier binärer Ganzzahlen, zum Beispiel:

^{1}	^{1}	^{1}	^{1}					(Übertrag)
	1	1	1	1	0	0	0	(Summand 1)
+	0	0	1	1	1	1	0	(Summand 2)
1	0	0	1	0	1	1	0	(Resultat)

Angenommen, die beiden Zahlen sind jeweils k Bit lang. Wenn eine der ganzen Zahlen weniger Bits als die andere hat, füllen wir mit führenden Bits wie in obigem Beispiel. Obwohl dieses Beispiel mit $k = 7$ Bit nur kleine Zahlen beschreibt, kann k sehr groß sein, beispielsweise 500 oder 1.000 Bit.

Analysieren wir nun im Detail, welche Einzelschritte diese Addition umfaßt. Grundlegend müssen die folgenden Schritte k mal ausgeführt werden:

① Lese das obere und das untere Bit und prüfe, ob es ein Übertrag-Bit gibt.

② Falls beide Bits 0 sind und es ist kein Übertrag vorhanden, schreibe eine 0 in die Resultatszeile und betrachte die nächste Spalte.

③ Falls entweder

(a) beide Bits sind 0 und es gibt ein Übertragsbit, oder

(b) eines der Bit ist 1 und es gibt kein Übertragsbit,

dann schreibe eine 1 in die Resultatszeile und gehe zur nächsten Spalte.

④ Falls entweder

(a) beide Bits sind 1 und es gibt kein Übertragsbit, oder

(b) eines der Bit ist 1 und es gibt ein Übertragsbit,

dann schreibe eine 0 als Ergebnis und eine 1 in den Übertrag der nächsten Spalte. Gehe zur nächsten Spalte.

⑤ Falls beide Bits 1 sind und das Übertragsbit ist ebenfalls 1, schreibe als Ergebnis eine 1 und in den Übertrag der nächsten Spalte ebenfalls eine 1. Weiter mit der nächsten Spalte.

Wird dieses Prozedere genau einmal ausgeführt, dann spricht man von einer **Bit-Operation**. Addiert man auf diese Weise zwei k-Bit Zahlen, dann erfordert dies k Bit Operationen. Wir werden sehen, dass komplexere Operationen ebenfalls auf Bit-Operationen zurückgeführt werden können. Die Zeit, die ein Computer benötigt, eine Aufgabe auszuführen, ist im wesentlichen proportional zu der Anzahl der Bit-Operationen. Die Proportionalitätskonstante – ein Bruchteil einer Nanosekunde pro Bit-Operation – hängt natürlich von dem speziellen Computersystem ab. Wenn man von der Zeit spricht, die benötigt wird, eine Aufgabe abzuarbeiten, versteht man darunter die Anzahl der erforderlichen Bit-Operationen zu bestimmen.

Daher ist die Zeit, die benötigt wird – i.e. die Anzahl der Bit-Operationen – um zwei ganze Zahlen zu addieren, gleich dem Maximum der Länge der beiden Zahlen. Wir schreiben dafür:

$$\text{Zeit } (k\text{-Bit} + l\text{-Bit}) = \max(k, l).$$

Will man die Zeit durch die beiden Zahlen ausdrücken, die addiert werden – sagen wir m und n – dann ist wegen $k = \text{Länge } (m) = O(\ln m)$

$$\text{Zeit}(m + n) = O(\max(\ln m, \ln n)).$$

Anmerkung:

Es ist ein erheblicher Unterschied, ob man die Laufzeit zur Durchführung einer Verarbeitung von ganzen Zahlen ausdrückt durch die ganzen Zahlen selbst (in diesem Beispiel m und n), oder durch die Länge dieser Zahlen (hier k und l). In Abhängigkeit von der Situation können beide Laufzeiten relevant sein, es ist jedoch wichtig, diese beiden Laufzeiten nicht zu verwechseln.

Wir untersuchen nun das Verfahren, eine k-Bit Zahl mit einer l-Bit Zahl zu multiplizieren, beispielsweise:

1	1	1	0	1	×	1	1	0	1
					1	1	1	0	1
			1	1	1	0	1	0	
		1	1	1	0	1			
	1	0	1	1	1	1	0	0	1

Allgemein nehmen wir an, wir verwenden dieses Verfahren, um eine k-Bit Zahl n mit einer l-Bit Zahl m zu multiplizieren. Dabei erhalten wir höchstens l Zeilen, wobei für jede 0 in der Zahl m eine Zeile weniger benötigt wird. Jede Zeile besteht aus einer Kopie von n, die nach links verschoben wird um eine bestimmte Distanz – i.e. mit 0en am rechten Ende angefügt. Um nun die benötigten Bit-Operationen zu zählen, nehmen wir an, dass wir jedesmal zwei Zeilen addieren, zuerst addiert man die zweite Zeile mit der ersten, dann die dritte Zeile mit dem Resultat der ersten Addition, dann addiert man die vierte Zeile mit dem Resultat der zweiten Addition usw. Mit anderen Worten, wir müssen höchsten $l - 1$ Additionen ausführen. In jeder Addition kopieren wir zuerst die rechts stehenden Bits von der oberen Zeile die über den mit 0en gefüllten Stellen der unteren Zeile stehen. Dieser Prozess – das einfache verschieben von Bits – zählt als *administrativer Vorgang* und wird nicht als Bit-Operation gezählt. Daher ignorieren wir diese Schritte in der Zeitbestimmung. Daher erfordert jede Addition nur k Bit-Operationen. Damit erhalten wir, dass die Gesamtzahl an Bit-Operationen für die Multiplikation einer k-Bit Zahl mit einer l-Bit Zahl weniger als

$$(l \text{ Additionen}) \times (k \text{ Bit-Operationen pro Addition}) = kl.$$

Anmerkungen:

1. Wir definieren die Zeit für die Durchführung einer arithmetischen Operation als obere Grenze für die Anzahl der Bit-Operationen, ohne spezielle Verfahren wie Shift-Operationen, Speicherzugriffe usw. zu betrachten.

2. Um eine Abschätzung des Zeitaufwands zu erhalten, die einfach und mit der bequem umzugehen ist, müssen wir an verschiedenen Stellen den *worst-case* annehmen. Beispielsweise können in einer Multiplikation viel weniger als $l-1$ Additionen von nichtverschwindenen Zeilen auftreten. Wenn wir aber bei der Bestimmung des O-Verhaltens eines Verfahrens sind, führt dies zu keiner Verbesserung.

3. Laufzeitbestimmungen haben nicht nur eine einzige richtige Antwort. Beispielsweise sind die folgenden Statements alle korrekt, wenn man die Laufzeitbestimmung der Multiplikation einer k-Bit Zahl mit einer l-Bit Zahl angibt:

   (a) Zeit $= O(kl)$

   (b) Zeit $< kl$

   (c) Zeit $< k(l-1)$

   (d) Falls die zweite Zahl die gleiche Anzahl von 0en und 1en hat ist Zeit $\leq kl/2$.

   Im folgenden werden wir die Big-O Notation für all diese Aussagen verwenden.

Die Laufzeitbestimmung für die Multiplikation kann auch ausgedrückt werden durch die Zahlen m und n selbst anstatt ihrer Längen:

$$\text{Zeit}(m \times n) = O(\ln m \ln n).$$

In dem speziellen Fall, dass die beiden Zahlen, die multipliziert werden, die gleiche Länge k haben, erhalten wir

$$\text{Zeit}(k \text{ Bit} \times k \text{ Bit}) = O(k^2).$$

Es sei an dieser Stelle angemerkt, dass in letzter Zeit viel Arbeit investiert wurde, Verfahren zu finden, um die Multiplikation zweier k-Bit Zahlen für großes k schneller durchführen zu können. Mathematiker haben ein Verfahren gefunden, welches für die Multiplikation zweier k-Bit Zahlen lediglich $O(k \ln k \ln \ln k)$ Bit-Operationen erfordert. Dies ist viel besser als $O(k^2)$ und sogar besser als $O(k^{1+\epsilon})$ für $\epsilon > 0$, egal wie klein ϵ gewählt wird.

21.2.2 Algorithmen

Wenn man allgemein die Anzahl von Bit-Operationen bestimmt, die zur Ausführung eines Verfahrens benötigt werden, besteht der erste Schritt darin, ein detaillierten Ablauf des Verfahrens aufzuschreiben. Wir haben dies oben für die Multiplikation durchgeführt. Ein explizites Schritt-für-Schritt Verfahren zur Durchführung einer Berechnung nennt man einen **Algorithmus**.

Es gibt natürlich verschiedene Algorithmen, die das gleiche Problem lösen, man kann das Verfahren wählen, das am einfachsten aussieht, oder man kann den schnellsten bekannten Algorithmus wählen, oder man wählt einen Kompromiss

und wählt einen Ausgleich zwischen Einfachheit und Schnelligkeit. Der obige Algorithmus für die Multiplikation von n mit m ist keinesfalls der schnellste bekannte Algorithmus, jedoch sicherlich um einiges schneller wie n-fache Addition von m mit sich selbst.

Bisher haben wir die binäre Addition und die binäre Multiplikation betrachtet. Die binäre Subtraktion funktioniert ähnlich wie die Addition. Wir haben das gleiche Laufzeitverhalten der Ordnung $O(k)$ für die Subtraktion zweier k-Bit Zahlen. Wir müssen für die Betrachtung der Subtraktion unsere Definition der Bit-Operation leicht erweitern. Eine Bit-Operation *Subtraktion* kann auf die gleiche Weise definiert werden wie die Bit-Operation *Addition* zuvor, wir müssen jedoch statt ein Bit in den Übertrag in die nächste Stelle zu schreiben, ein Bit von der nächsten Stelle 'borgen'. Außerdem führt die Betrachtung der Subtraktion zu einer anderen Liste von vier Alternativen.

Beispiel:

Wir betrachten die (binäre) Subtraktion der beiden Zahlen `110 1101` ($k = 7$ Bit) und `1 1010` ($l = 4$ Bit). Nach der Schulmethode erhalten wir:

	1	1	0	1	1	0	1	(Minuend)
(-)	0	0	1	1	0	1	0	(Subtrahend)
		(1)			(1)			(Borgen-Stelle)
	1	0	1	0	0	1	1	

Die Logik der Subtraktion kann durch folgende Tabelle dargestellt werden.

Minuend	**Subtrahend**	**Borgen**	**Resultat**	**Borgen n. Stelle**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Gehen wir das Verfahren nun Schritt für Schritt durch.

① Lese das Minuend-Bit (oberes Bit) und das Subtrahend-Bit (unteres Bit) von rechts nach links und prüfe, ob es ein 'Borgen'-Bit gibt.

② Falls

(a) beide Bits sind 0 und es gibt kein 'Borgen' Bit – oder

(b) beide Bits sind 1 und es gibt kein 'Borgen' Bit – oder

(c) das 'Borgen' Bit ist 1, das Subtrahend-Bit ist 0 und das Minuend-Bit ist 1, dann

schreibe eine 0 und gehe zur nächsten Stelle.

③ Falls

(a) das Minuend-Bit 0 ist, das Subtrahend-Bit ist 0 und das 'Borgen'-Bit ist 1 oder

(b) das Minuend-Bit ist 0, das Subtrahend-Bit ist 1 und das 'Borgen'-Bit ist 0 oder

(c) das Minuend-Bit ist 1, das Subtrahend-Bit ist 1 und das 'Borgen'-Bit ist 1

schreibe eine 1 in das Resultat und eine 1 in die 'Borgen'-Stelle der nächsten Spalte, gehe zur nächsten Stelle.

④ Falls das Minuend-Bit 0, das Subtrahend-Bit 1 und das 'Borgen'-Bit 1 ist, schreibe eine 0 in das Resultat und eine 1 in die 'Borgen'-Stelle' der folgenden Spalte.

⑤ Falls das Minuend-Bit 1 ist, das Subtrahend- und das 'Borgen'-Bit 0 ist, schreibe eine 1 in das Resultat, gehe zur nächsten Spalte.

Aus dieser expliziten Form des Algorithmus für die Subtraktion erkennen wir, dass für die Subtraktion einer l-stelligen Zahl von einer k-stelligen Zahl mit $l < k$ ebenfalls k Bit-Operationen durchgeführt werden müssen, daher hat die Subtraktion ebenfalls den Zeitaufwand $O(k)$.

Die binäre Division kann auf die gleiche Weise analysiert werden wie die Multiplikation. Das Resultat ist, dass $O(l(k-l+1))$ Bit-Operationen benötigt werden, um den Quotienten und den Rest zu erhalten, wenn eine k-Bit Zahl durch eine l-Bit Zahl dividiert wird, wobei $k \geq l$.

Beispiel:

Wir betrachten die Division der binären Zahl `111 0101` ($k = 7$ Bit) – dies ist der Dividend – durch die binäre Zahl `1 010` mit $l = 4$ Bit, dies ist der Teiler oder Divisor.

	1	1	1	0	1	0	1	:	1	0	1	0	= 1 0 1 1
(-)	1	0	1	0									
		1	0	0	1	0							
	(-)		1	0	1	0							
			1	0	0	0	1						
		(-)		1	0	1	0						
			0	0	1	1	1						

Damit erhalten wir das Resultat:

`111 0101 : 1 010 = 1 011 Rest 111`

oder im Dezimalsystem: 117 : 10 = 11 Rest 7.

Die Vorgehensweise bei der Anwendung der Schulmethode für die Division kann durch folgende Schritte beschrieben werden:

① Schreibe den l-Bit Teiler (`1 010`) von links unter den Dividenden (`111 0101`).

② Prüfe, ob der l-Bit Anteil des Dividenden größer oder gleich dem Teiler ist. Wenn nein, verschiebe den Teiler um eine Stelle nach rechts. Sonst, tue nichts.

③ Schreibe eine 1 in den Quotienten.

④ Subtrahiere den Teiler vom l-Bit Anteil bzw. $l+1$-Bit Anteil des Dividenden.

⑤ Schreibe das Resultat r unter den Teiler.

⑥ Füge die nächste Stelle des Dividenden an das Resultat, dies führt auf r'.

⑦ Prüfe, ob der Teiler größer oder gleich r' ist. Wenn ja, füge so lange weitere Stelle des Dividenden an r' an, dies führt auf r'', bis Teiler kleiner als r'' ist. Schreibe für jede angefügte Stelle eine 0 in den Quotienten an die rechte Stelle.

⑧ Wenn Teiler kleiner als r'', füge eine 1 an die rechte Stelle des Quotienten an und subtrahiere Teiler von r''.

⑨ Wiederhole bis keine Stellen aus dem Dividenden mehr vorhanden sind.

Betrachten wir ein weiteres Beispiel:

```
      1  1  1  1  1  1  1  :  1  0 = 1 1 1 1 1 1
(-)   1  0
      ---------
         1  1
   (-)   1  0
         ------
            1  1
      (-)   1  0
            ------
               1  1
         (-)   1  0
               ------
                  1  1
            (-)   1  0
                  ------
                     1  1
               (-)   1  0
                     ------
                        1
```

An diesem Beispiel erkennt man folgendes: Der Dividend ist die $k = 7$-Bit Zahl `1111111`, der Divisor die $l = 2$-Bit Zahl `10`. Es sind insgesamt 6 Subtraktionen erforderlich, bis der Quotient und der Rest berechnet sind. Dies ist genau der Extremfall mit $k - l + 1$ Subtraktionen. Jede Subtraktion selbst erfordert den Zeitaufwand $O(l)$, daher führt die Division im worst-case auf einen Zeitaufwand von $O(l(k - l + 1))$.

Beispiel:

In einem weiteren Beispiel bestimmen wir den Zeitaufwand, um eine k-Bit Integerzahl n in ihre Darstellung im Dezimalsystem umzurechnen.

Der Algorithmus zur Umrechnung funktioniert folgendermaßen.

- Dividiere n durch $\texttt{1 010}_2$. Der Rest – das ist eine der ganzen Zahlen `0, 1, 10, 01, 11, 100, 101, 110, 111, 1000` oder `1001` – ist die Einerziffer d_0.
- Ersetze n durch den Quotienten und wiederhole den Prozess, i.e. dividiere den Quotienten durch $\texttt{1 010}_2$. Der Rest dieser Division ist die Dezimalziffer d_1, die Zehnerziffer, der Quotient ist die Zahl, die im nächsten Schritt durch $\texttt{1 010}_2$ dividiert wird.
- Diese Schritte sind so oft zu wiederholen, wie die Anzahl der Dezimalstellen von n ist, dies ist

$$\lfloor \frac{\ln n}{\ln 10} \rfloor + 1 = O(k).$$

Betrachte als Beispiel die Umwandlung der binären Zahl

$$n = \texttt{10110011101}_2$$

in das Dezimalsystem.

Schritt 1: Dividiere n durch $\texttt{1 010}_2 = 10_{10}$:

```
    1  0  1  1  0  0  1  1  1  0  1  :  1 010  =  1000 1111
(-) 1  0  1  0
    ---------
          1  0  0  1  1
       (-)   1  0  1  0
       ---------------
          1  0  0  1  1
       (-)   1  0  1  0
       ---------------
             1  0  0  1  0
          (-)   1  0  1  0
          ---------------
                1  0  0  0  1
             (-)   1  0  1  0
             ---------------
                   0  0  1  1  1
```

Also ist $d_0 = \texttt{111}_2 = 7_{10}$.

Schritt 2: Dividiere den Quotienten `1000 1111` durch $\texttt{1 010}_2 = 10_{10}$:

```
      1  0  0  0  1  1  1  1  :  1 010  =  1 110
(-)      1  0  1  0
      ------------
         1  1  1  1
   (-)   1  0  1  0
   ---------------
            1  0  1  1
         (-)   1  0  1  0
         ---------------
               1  1
```

In diesem Schritt haben wir also $d_1 = \texttt{11} = 3_{10}$.

Schritt 3: Dividiere den Quotienten `1 110` durch `1 010`:

$$\begin{array}{rcccccccccc} & 1 & 1 & 1 & 0 & : & 1 & 0 & 1 & 0 & = 1 \\ (\text{-}1) & 1 & 0 & 1 & 0 & & & & & & \\ \hline & & 1 & 0 & 0 & & & & & & \end{array}$$

Damit haben wir im dritten Schritt die dritte Ziffer erhalten: $d_3 = 100_2 = 4_{10}$.

Schritt 4: Schließlich bleibt:

$$1 \ : \ 1010 = 0 \text{ Rest } 1$$

Damit ist $d_4 = 1$.

Zusammengefasst haben wir als:

$$n = 10110011101_2 = 1437_{10}.$$

Wir bestimmen nun, wie viele Bit-Operationen diese Berechnungen erfordern. Wir haben $O(k)$ Divisionen, um die Ziffern $d_0, d_1, d_2, \ldots$ zu finden. Jede Division erfordert $O(4k)$ Bit-Operationen, denn die Division einer k-Bit Zahl durch eine 4-Bit Zahl erfordert maximal $4k$ Bitoperationen. Da aber $O(4k)$ das gleiche ist wie $O(k)$ in der Big-O Notation, dann folgern wir, dass die Gesamtzahl an Bit-Operationen für die Umrechnung einer Binärzahl in Dezimalzahldarstellung

$$O(k) \cdot O(k) = O(k^2)$$

Bit-Operationen erfordert. Drückt man dies durch die Zahl n selbst aus und nicht durch deren Länge im Binärsystem, dann erhalten wir mit $k = O(\ln n)$:

$$\text{Zeit, } n \text{ in Dezimalzahl umzuwandeln: } = O(\ln^2 n).$$

Im nächsten Beispiel bestimmen wir den Zeitaufwand, eine k-Bit Zahl n in die Darstellung zu einer beliebigen Basis b umzuwandeln. b kann dabei eine sehr große Zahl sein.

Lösung:

Wir verwenden den gleichen Algorithmus wie im vorherigen Beispiel mit der Ausnahme, dass nun statt durch die 4 Bit Zahl 1 010 durch eine l-Bit Zahl b dividiert werden muß. Wenn l groß genug ist, erfordert dies natürlich mehr Zeit als vorher. In diesem Fall sind dies $O(kl)$ Bit-Operationen. Die Anzahl der Divisionen erhält man, indem man die Anzahl der Basis-b Stellen der Zahl n bestimmt, das sind aber gerade $O(k/l)$ Stellen. Dies entspricht genau der Anzahl der Divisionen mit Sammeln der Reste, um die Basis b Darstellung zu finden. Insgesamt hat man daher

$$O(k/l) \cdot O(k) = O(k^2)$$

Bit-Operationen durchzuführen. Wir erhalten daher die gleiche Anzahl an Operationen wie bei der Umrechnung einer Binärzahl in die 10er Basis, i.e. der Zeitaufwand zur Umwandlung einer Binärzahl in die Darstellung in einer Basis b hängt nicht von der Basis b ab.

Beispiel:

Man bestimme den Zeitaufwand zur Berechnung von $n!$.

Lösung:

Wir wenden zur Berechnung den folgenden Algorithmus an. Wir multiplizieren zunächst 2 mit 3, dann das Resultat mit 4, dann dieses Resultat mit 5 usw. bis man bei n angelangt ist. Im $j-1$ten Schritt multipliziert man also $j!$ mit $j+1$. Bei diesem Verfahren hat man also insgesamt $n-2$ Multiplikationen, wobei jede Multiplikation darin besteht, ein Teilprodukt (nämlich $j!$) mit der nächsten ganzen Zahl zu multiplizieren. Das Teilprodukt wird sehr groß werden. Für eine worst-case Laufzeitbestimmung für die Anzahl der Bits, die hierbei auftreten, nehmen wir die Anzahl der Bits im letzten Produkt, i.e. in $n!$. Nun ist

$$\text{Länge } (n!) \leq n \cdot \text{ Länge } (n) = O(n \ln n).$$

In jeder der $n-2$ Multiplikationen bei der Berechnung von $n!$ multiplizieren wir eine ganze Zahl mit höchstens $O(\ln n)$ Bits (nämlich $j-1$) mit einer ganzen Zahl mit $O(n \ln n)$ Bits – das ist $j!$. Dies erfordert $O(n \ln^2 n)$ Bit-Operationen. Dies müssen wir $n-2$ mal ausführen, also $O(n)$ mal. Daher ist die Gesamtzahl an Bit-Operationen zur Berechnung von $n!$

$$O(n \ln^2 n) \cdot O(n) = O(n^2 \ln^2 n).$$

Damit erhalten wir:

$$\text{Laufzeit zur Berechnung von } n! = O(n^2 \ln^2 n).$$

21.3 Laufzeitbestimmung des Euklidischen Algorithmus

Gegeben sind zwei ganze Zahlen $0 < b < a$. Der größte gemeinsame Teiler d von a und b kann berechnet werden und die Gleichung

$$au + bv = d$$

ist nach den ganzen Zahlen u, v auflösbar in der Laufzeit $O(\ln a \cdot \ln b)$.

Lösung:

Wir wenden den erweiterten EUKLIDischen Algorithmus an. Zunächst dividieren

wir sukzessive:

$$\begin{aligned}
a &= q_0 b + r_1 && 0 < r_1 < b \\
b &= q_1 r_1 + r_2 && 0 < r_2 < r_1 \\
r_1 &= q_2 r_2 + r_3 && 0 < r_3 < r_2 \\
&\vdots \quad \vdots && \vdots \\
r_{j-1} &= q_j r_j + r_{j+1} && 0 < r_{j+1} < r_j \\
&\vdots \quad \vdots && \vdots \\
r_{l-2} &= q_{l-1} r_{l-1} + r_l && 0 < r_l < r_{l-1} \\
r_{l-1} &= q_l r_l + r_{l+1} && 0 < r_{l+1} < r_l \\
r_l &= q_{l+1} r_{l+1}
\end{aligned}$$

daher ist $d = r_{l+1}$. Nun rechnen wir rückwärts:

$$\begin{aligned}
d &= r_{l+1} \\
&= r_{l-1} - q_l r_l \\
&= v_l r_l + u_{l-1} r_{l-1} && v_l = -q_l && u_{l-1} = 1 \\
&= v_{l-1} r_{l-1} + u_{l-1} r_{l-1} && v_{l-1} = u_{l-1} - q_{l-1} v_l && u_{l-2} = v_l \\
&\quad \vdots \\
&= v_j r_j + u_{j-1} r_{j-1} && v_j = u_j - q_j v_{j+1} && u_{j-1} = v_{j+1} \\
&\quad \vdots \\
&= v_1 r_1 + u_0 b && v_1 = u_1 - q_1 v_2 && u_0 = v_2 \\
&= vb + ua && v = u_0 - q_0 v_1 && u = v_1
\end{aligned}$$

Beispiel:

Sei $a = 5060$ und $b = 322$. Dann folgt durch Anwendung des obigen Algorithmus:

$a = q_0 b + r_1$	$5060 = 15 \cdot 322 + 230$	$q_0 = 15, r_1 = 230$
$b = q_1 r_1 + r_2$	$322 = 1 \cdot 230 + 92$	$q_1 = 1, r_2 = 92$
$r_1 = q_2 r_2 + r_3$	$230 = 2 \cdot 92 + 46$	$q_2 = 2, r_3 = 46$
$r_2 = q_3 r_3 + r_4$	$92 = 2 \cdot 46 + 0$	$q_3 = 2, r_4 = 0.$

Damit ist für $l = 2$

$$d = \mathrm{ggT}(5060, 322) = r_3 = 46.$$

Rechnet man rückwärts, so erhält man:

$$\begin{aligned} d &= r_3 \\ &= r_1 - q_2 r_2 \\ &= r_1 - q_2(b - q_1 r_1) \\ &= (a - q_0 b) - q_2\big(b - q_1(a - q_0 b)\big) \\ &= (1 + q_1 q_2)a - (q_0 + q_2 + q_0 q_1 q_2)b. \end{aligned}$$

Einsetzen der Wert ergibt:

$$d = 3 \cdot 5060 - 47 \cdot 322.$$

In der obigen Notation erhalten wir auf die gleiche Weise:

$$\begin{aligned} d &= r_3 \\ &= r_1 - q_2 r_2 \\ &= v_2 r_2 + u_1 r_1 && \text{mit } v_2 = -q_2, u_1 = 1 \\ &= v_2(b - q_1 r_1) + u_1 r_1 \\ &= (u_1 - q_1 v_2) r_1 + v_2 b \\ &= v_1 r_1 + u_0 b && \text{mit } v_1 = u_1 - q_1 v_2, u_0 = v_2 \\ &= v_1(a - q_0 b) + u_0 b \\ &= va + ub && \text{mit } v = v_1, u = v_0 - v_1 q_0. \end{aligned}$$

Beispiel:

Sei $a = 2145$ und $b = 238$. Wir erhalten:

$a = q_0 b + r_1$	$2145 = 9 \cdot 238 + 3$	$q_0 = 9, r_1 = 3$
$b = q_1 r_1 + r_2$	$238 = 79 \cdot 3 + 1$	$q_1 = 79, r_2 = 1$
$r_1 = q_2 r_2 + r_3$	$3 = 3 \cdot 10$	$q_2 = 3, r_3 = 0$

Damit ist

$$d = \mathrm{ggT}(2145, 238) = r_2 = 1.$$

Das heißt, $a = 2145$ und $b = 238$ sind coprim. Rechnen wir rückwärts, dann erhalten wir:

$$\begin{aligned} d &= 1 \\ &= r_2 \\ &= b - q_1 r_1 \\ &= b - q_1(a - q_0 b) \\ &= (1 + q_0 q_1)b - q_1 a \\ &= (1 + 9 \times 79) \cdot 238 - 79 \cdot 2145 \\ &= 712 \cdot 238 - 79 \cdot 2145. \end{aligned}$$

Mit anderen Worten,

$$712 \cdot 238 = 79 \cdot 2145 + 1$$

oder

$$712 \cdot 238 \equiv 1 \bmod 2145,$$

das heißt, 712 ist das multiplikative Inverse von 238 modulo 2145.

Um den Zeitaufwand zu erhalten, um der erweiterten EUKLIDischen Algorithmus zu durchlaufen, erinnern wir uns, dass die Anzahl der Bit-Operationen bei der Division

$$a = q_0 b + r_1$$

von der Ordnung Länge$(b) \cdot$ Länge(q_0) ist. Analog ist der Zeitaufwand für die Division

$$r_{j-1} = q_j r_j + r_{j+1}$$

höchstens

$$\text{Länge } (q_j) \cdot \text{ Länge } (r_j) \leq \text{ Länge } (b) \cdot \text{ Länge } (q_j).$$

Damit ist der Gesamtaufwand für alle Divisionen

$$O\big(\ln b \cdot (\ln q_0 + \ln q_1 + \ldots + \ln q_{l+1})\big) = O\big((\ln b) \cdot (\ln \prod q_j)\big)$$

Nun ist leicht zu sehen, dass

$$\prod q_j \leq a,$$

denn

$$a = q_0 \cdot b + r_1 > q_0 \cdot b, \text{ da } 0 < r_1 < b.$$

Setzt man hier sukzessive

$$\begin{aligned} a &= q_0 b + r_1 \\ &= q_0(q_1 r_1 + r_2) + r_1 \\ &= q_0 q_1 r_1 + q_0 r_2 + r_1 \\ &= q_0 q_1 (q_2 r_2 + r_3) + q_0 r_2 + r_1 \\ &= q_0 q_1 q_2 r_2 + q_0 q_1 r_3 + q_0 r_2 + r_1 \\ &\vdots \\ &= q_0 q_1 q_1 \cdots q_{j-1}(q_j r_j + r_j + 2) + \cdots \end{aligned}$$

dann erkennt man, dass gilt:

$$a \geq \prod q_j.$$

Damit erhalten wir für die Laufzeit der Berechnung des ggT die obere Schranke

$$O(\ln b \cdot \ln a).$$

Um den EUKLIDischen Algorithmus rückwärts zu rechnen, i.e. die Berechnung aller

$$v_j = u_j - q_j v_{j+1}$$

benötigt man ebenfalls den Zeitaufwand $O(\ln a \ln b)$. Daher ist der Zeitaufwand des erweiterten EUKLIDischen Algorithmus $O(\ln^2 a)$.

Anmerkung:

Eine unmittelbare Konsequenz aus diesem Beispiel ist, dass das multiplikative Inverse einer Zahl a modulo einer ganzen Zahl m mit dem Aufwand $O(\ln^2 m)$ mit Hilfe des erweiterten EUKLIDischen Algorithmus berechnet werden kann. Das heißt als, jede Kongruenz der Form

$$a \cdot x \bmod m$$

mit $\mid a \mid < m$ und $\text{ggT}(a, m) = 1$ kann nach x mit dem Aufwand $O(\ln^2 m)$ gelöst werden.

Beispiel:

Angenommen, m ist eine natürliche k-Bit Zahl, N ist eine natürliche l-Bit Zahl und $\mid b \mid < m$. Die kleinste nicht-negative Restklasse von

$$x = b^N \bmod m$$

kann in der Zeit $O(k^2 l)$ bestimmt werden.

Lösung:

Die Berechnung der kleinsten Restklasse geschieht durch das *wiederholtes Quadrieren* Verfahren, das die Grundlage des `fastexp` Algorithmus ist[2]. Wir schreiben zunächst N in Binärform:

$$N = \sum_{i=0}^{l-1} b_i 2^i = b_{l-1} 2^{l-1} + b_{l-2} 2^{l-2} + \ldots + b_2 2^2 + b_1 2^1 + b_0$$

[2] Dieses Verfahren nennt man auch *Quadrieren und Multiplizieren* Methode.

und setzen:

$$\begin{aligned} x &= b^N \bmod m \\ &= b^{\sum b_i 2^i} \bmod m \\ &= \Big(\prod_{i=0}^{l-1} b^{b_j 2^j} \bmod m\Big) \bmod m \end{aligned}$$

Wir berechnen nun sukzessive die kleinste nichtnegative Restklasse von

$$b^{2^k} \text{ für } k = 1, 2, \ldots, l-1.$$

Um b^{2^k} zu berechnen, benutzen wir den zuvor berechneten Wert $b^{2^{k-1}}$, quadrieren diesen und reduzieren dann modulo m. Da keien der Zahlen, mit denen wir arbeiten, eine größere Länge hat als $2k$ (denn multipliziert man zwei Zahlen in $\mathbb{Z}_m$ erhält man eine Zahl kleiner m^2), benötigt dieses Verfahren den Zeitaufwand $O(k^2)$ für jedes j.

Seien nun $j_1, j_2, \ldots, j_\lambda$ die Indizes, für die $b_{j_\nu} = 1$, i.e. die Positionen der 1er Bits in der Binärdarstellung von N. Dann ist

$$N = \sum 2^{j_\nu}$$

und

$$b^N = \prod b^{2^{j_\nu}}.$$

Wir multiplizieren zunächst die kleinste nichtnegative Restklasse von $b^{2^{j_1}}$ mit der kleinsten, nichtnegativen Restklasse von $b^{2^{j_2}}$ und reduzieren das Resultat modulo m. Wir multiplizieren das Resultat dieser Operation mit der kleinsten nichtnegativen Restklasse von $b^{2^{j_3}}$ und reduzieren wieder modulo m usw. Das Endergebnis ist b^N. Daraus folgt der Zeitaufwand von $O(k^2 l)$ für diesen Algorithmus.

21.3.1 Von polynomialer zur exponentieller Laufzeit

Wir geben nun eine fundamentale Definition bei der Untersuchung von Algorithmen.

Definition

Ein Algorithmus zur Ausführung einer Berechnung heißt Polynomialzeit Algorithmus, wenn es eine ganze Zahl d gibt, so dass die Anzahl der Bit-Operationen zur Ausführung des Algorithmus auf ganze Zahlen der Länge k höchstens $O(k^d)$ ist.

Die üblichen arithmetischen Operationen $+, -, \times, \div$ sind also Beispiele für Polynomialzeit Algorithmen, ebenfalls die Umwandlung von einer Zahlenbasis zu einer anderen. Andererseits erfordert die Berechnung von $n!$ den Zeitaufwand $O(n^2 \ln^2 n)$ und ist daher kein Polynomialzeit Algorithmus.

Anmerkung:

Das Statement *Ausführung des Algorithmus auf ganze Zahlen der Länge k* in der obigen Definition ist etwas vage. Was gemeint ist, ist der folgende Sachverhalt: Wenn eine Berechnung durchgeführt wird, sollte immer auch die Form des Inputs festgelegt sein. Dann bedeutet k in der obigen Definition die gesamte binäre Länge des Inputs. In vielen Problemen ist die Form der Eingabe offensichtlich und wird üblicherweise nicht explizit angegeben. Das folgende Beispiel zeigt, dass man manchmal sehr sorgfältig mit dieser Formulierung umgehen muß.

Beispiel:

Gibt es eine Polynomialzeit Algorithmus um zu bestimmen, ob die m-te FERMAT Zahl eine Primzahl oder zusammengesetzt ist?

In diesem Beispiel ist es entscheidend, die Art des Inputs zu spezifizieren. Ist der Input die Zahl

$$n = 2^{2^m} + 1$$

in Binärdarstellung – i.e.

$$n = 1\underbrace{000\ldots00}_{2^m-1}1$$

dann ist die Antwort auf die obige Frage sicherlich 'ja'. Das heißt, es gibt mehrere Algorithmen, die mit eine Laufzeit, die beschränkt ist durch eine polynomiale Funktion von 2^m bestimmen können, ob n eine Primzahl ist oder zusammengesetzt ist. Ist jedoch der Input die Zahl m in Binärdarstellung, dann ist die Antwort auf die obige Frage mit Sicherheit 'nein', denn es ist kein Algorithmus bekannt, der die Primzahleigenschaft der m-ten FERMATzahl in eine Laufzeit feststellen kann, die beschränkt ist durch eine feste Potenz von $\log_2 m$.

Eine Klasse von Algorithmen, die weit von der Polynomialzeit ist, ist die Klasse der **Exponentialzeit Algorithmen**. Solche Algorithmen haben eine Laufzeit der Form $O(e^{ck})$, wobei c eine Konstante ist. Hierbei ist k die Gesamtlänge der ganzen Zahlen (in Binärdarstellung), auf die der Algorithmus angewendet wird.

Beispiel:

Der Faktorisierungsalgorithmus 'Teilermethode' aus Abschnitt [19.1.2] benötigt die Laufzeit $O(n^{1/2+\epsilon})$, mit $\epsilon > 0$, beliebig klein. Da

$$k \approx \log_2 n = \frac{\ln n}{\ln 2}.$$

Daher ist

$$n^{1/2+\epsilon} = e^{(1/2+\epsilon)\ln 2 \cdot k} = e^{ck}, c = (\frac{1}{2} + \epsilon) \cdot \ln 2,$$

und die Laufzeit des Algorithmus 'Teilermethode' kann daher ausgedrückt werden in der Form $O(e^{ck})$.

Es gibt einen zweckmäßigen Ausdruck, um Laufzeiten zwischen polynomialer und exponentialer Laufzeit zu klassifizieren. Sei n eine große positve ganze

Zahl, i.e. der Input eines Algorithmus. Sei γ eine Zahl zwischen 0 und 1; und sei $c > 0$ eine Konstante.

Definition:

$$L_n(\gamma; c) = O\Big(e^{c\left((\ln n)^{\gamma}(\ln\ln n)^{1-\gamma}\right)}\Big).$$

Insbesondere erhalten wir:

$$L_n(1; c) = O(e^{c \ln n}) = O(n^c),$$
$$L_n(0; c) = O(e^{c \ln\ln n}) = O\big((\ln n)^c\big).$$

Ein $L(\gamma)$-Algorithmus ist ein Algorithmus, der – angewendet auf eine ganze Zahl n – eine Laufzeit der Form $L_n(\gamma; c)$ hat für ein c. Insbesondere ist ein **Polynomialzeit Algorithmus** ein $L(1)$ Algorithmus und ein **Exponentialzeit Algorithmus** ist ein $L(0)$ Algorithmus. Unter einem **Subexponentialzeit Algorithmus** verstehen wir eine $L(\gamma)$ Algorithmus für ein $\gamma < 1$.

Grob gesprochen ist der Parameter γ ein Maß dafür, wie nahe die Laufzeit eines Algorithmus bei polynomialem oder exponentiellem Verhalten liegt. Wir haben oben gesehen, dass das naive Teilerverfahren zur Faktorisierung einer Zahl n ein $L(1)$ Algorithmus ist. Bis vor kurzem waren die besten (probabilistischen) Faktorisierungsalgorithmen alle $L(1/2)$-Algorithmen. Mit dem Aufkommen der Zahlenfeld-Siebe[3] wurde diese obere Schranke auf $L(1/3)$ heruntergeschraubt.

[3]Siehe A.K. Lenstra und H.W. Lenstra Jr., *The Development of the Number Field Sieve*, Lecture Notes in Mathematics, **1554**, (1993), Springer.

21.4 $\mathcal{P}$, $\mathcal{NP}$ und $\mathcal{NP}$ vollständige Probleme

In diesem Kapitel betrachten wir drei fundamentale Begriffe aus der Computerwissenschaft

- die Klasse $\mathcal{P}$ der Entscheidungsprobleme, die in polynomialer Laufzeit lösbar sind
- die Klasse $\mathcal{NP}$ der Entscheidungsprobleme mit nichtdeterministischer polynomialer Laufzeit
- die Klasse der $\mathcal{NP}$-vollständigen Probleme; dies sind Entscheidungsprobleme, die vollständig sind.

Wir geben hier eine informelle Einführung in dieses Themengebiet.

21.4.1 Instanzen von Problemen, Suchprobleme und Entscheidungsprobleme

Unter dem Begriff *Problem* verstehen wir hier die allgemeine Beschreibung einer Aufgabe, der Ausdruck *Instanz* eines Problems bedeutet einen speziellen Fall dieser Aufgabe.

Beispiel:

Das **Suchproblem der Faktorisierung ganzer Zahlen** ist das Problem, einen nichttrivialen Faktor p einer gegebenen ganzen Zahl n zu finden oder sonst festzustellen, dass es keinen nichttrivialen Faktor gibt, i.e. die Zahl n ist Primzahl. Haben wir eine konkrete Zahl für n gegeben, dann liegt eine Instanz des Suchproblems der Faktorisierung ganzer Zahlen vor.

Beispiel:

Das **Traveling Salesman Problem** (TSP) ist die Aufgabe den kürzesten Weg einer Rundreise zu finden. Der Salesman startet in einer Stadt A und besucht alle Städte auf seiner Liste genau ein Mal und kehrt wieder zu der Stadt A zurück. Eine *Instanz* des TSP bestht in einer speziellen Liste von Städten und die Distanzen zwischen allen Paaren von Städten auf dieser Liste.

Beispiel:

Das 3-Farben Problem ist die Aufgabe, eine gegebene Landkarte mit drei Farben zu erstellen, so dass keine zwei benachbarte Regionen die gleiche Farbe haben (wenn dies möglich ist). Üblicherweise reduziert man das Problem auf die Färbung eines Graphen. Das 3-Farben Problem für Graphen besteht in der Aufgabe, eine von drei Farben jedem Vertex zuzuordnen, so dass keine verbundenen Vertexpaare die gleiche Farbe haben (siehe Abbildung [21.1]).

Der Begriff *Input* bezoht sich auf alle Informationen, die spezifiziert werden müssen, um eine Instanz eines Problems beschreiben zu können. Bei der Faktorisierung ganzer Zahlen ist der Input einfach die zu faktorisierende Zahl N. Der Terminus *Input Länge* bezieht sich auf die Anzahl der Symbole, die benötigt werden, um den Input aufzulisten. Dieses spezielle System von Symbolen wird

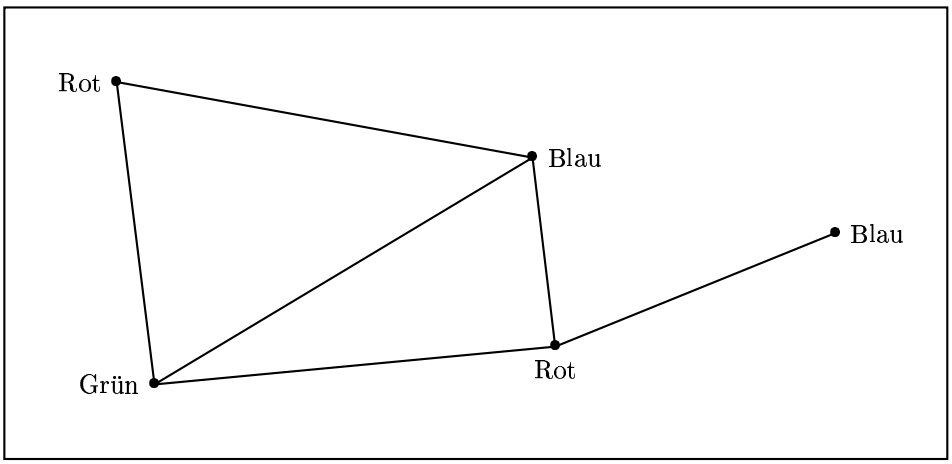


Abbildung 21.1: Beispiel eines Graphen, der mit drei Farben einfarbbar ist

einfach einmal festgelegt. Schreiben wir beispielsweise die Zahlen in Binärdarstellung, dann ist die Input Länge bei einer Instanz des Faktorisierungsproblems $1 + \lfloor \log_2 N \rfloor$.

Im Traveling Salesman Problem besteht der Input aus m Städten. Wenn wir annehmen, dass die Städte von 1 bis m durchnummeriert sind, dann ist der Input eine spezielle Abbildung von der Menge der Paare $(i, j), 1 \leq i < j \leq m$ in die Menge der nnatürlichen Zahlen $\mathbb{N}$.

In einem 3-Farben Problem mit m Vertices – wobei wir annehmen, dass die vertices von 1 bis m durchnummeriert sind – kann man den Input auffassen als Teilmenge der Menge von Paaren $(i, j), 1 \leq i < j \leq m$. Das bedeutet, der input ist ein Graph $G = (V, E)$, wobei V die Vertexmenge $V = \{1, 2, \ldots, m\}$ und $E \subset \{(i, j)\}_{1 \leq i < j \leq m}$ die Menge der Kanten bezeichnet.

Um die Definitionen von $\mathcal{P}$ und $\mathcal{NP}$ zu geben, müssen wir unsere Probleme leicht modifizieren, so dass man **Entscheidungsprobleme** untersucht. Ein Entscheidungsproblem ist ein Problem, dessen Ausgabe aus einer Ja-oder-Nein Antwort besteht. Ist andererseits die gewünschte Ausgabe mehr als 'Ja' oder 'Nein' – i.e. will man eine Zahl erhalten oder eine Reiseroute usw. – dann nennt man das Problem ein **Suchproblem**.

Anmerkung:

Im Gegensatz zu Entscheidungsproblemen kann ein Suchproblem mehrere Lösungen haben. Beispielsweise sucht man im Traveling Salesman Problem einen Rundweg mit minimale Länge, der durch alle Städte geht. Es kann durchaus möglich sein, dass es mehrere solcher minimaler Rundwege gibt.

Beispiel:

Eine Instanz der Faktorisierung ganzer Zahlen als Entscheidungsproblem lautet folgendermaßen:

`INPUT`: Positive ganze Zahlen N und k.

FRAGE: Hat N einen Faktor M mit $2 \leq M \leq k$?

Das eigentliche Problem, einen nichttrivialen Faktor von M von N zu finden, nennt man das Faktorisierungs Suchproblem.

Beispiel:

Eine Instanz des Traveling Salesman Entscheidungsproblems hat die Form:

INPUT: Eine ganze Zahl m, eine Abbildung von der Menge der Paare $(i,j), 1 \leq i < j \leq m$ in die natürlichen Zahlen und eine ganze Zahl k.

FRAGE: Gibt es eine Tour durch die Städte der Länge $\leq k$?

Das Traveling Salesman Suchproblem ist das Problem, eine konkrete Tour mit minimaler Länge zu finden.

Beispiel:

Eine Instanz des 3-Farben Entscheidungsproblems hat die Form:

INPUT: Ein Graph $G = (V, E)$

FRAGE: Hat dieser Graph eine 3-Färbung? Mit anderen Worten, gibt es eine Abbildung c von V auf eine 3-elementige Menge so, dass $(i,j) \in E \implies c(i) \neq c(j)$?

Für viele Probleme – einschließlich Faktorisierung, Traveling Salesman und 3-Farben – sind die Entscheidungsprobleme und die Suchprobleme im wesentlichen äquivalent. Das bedeutet, dass ein Algorithmus, der das eine Problem löst leicht in einen Algorithmus umgewandelt werden kann, der das andere Problem löst.

Beispiel:

Sehen wir uns an, wie dies beim Faktorisierungsproblem durchgeführt wird.

- Wir nehmen zunächst an, wir haben einen Algorithmus für das Suchproblem. Das bedeutet, bei gegebenem N wenden wir den Algorithmus an und finden eine nichttrivialen Faktor M. Dann wenden wir den Algorithmus erneut um nichttriviale Faktoren von M und N/M zu erhalten usw. Dies wird solange fortgeführt, bis die ganze Zahl M in Potenzen von Primfaktoren zerlegt ist. Verfügen wir über die Primfaktorzerlegung der Zahl N, können wir sofort bestimmen, ob N einen Faktor im Intervall $[2,k]$ hat oder nicht. Die Antwort auf diese Frage ist JA, dann und nur dann, wenn der kleinste Primteiler von N in diesem Intervall liegt.
- Umgekehrt nehmen wir nun an, wir haben einen Algorithmus, der das Entscheidungsproblem durchführt. In diesem Fall können wir den binären Suchalgorithmus verwenden, um den exakten Wert des Faktors zu erhalten, gleichzeitig lösen wir das Faktorisierungs Suchproblem. Präziser: Wir finden einen nichttrivalen Faktor von N Bit für Bit, indem wir mit dem Führenden Bit beginnen. Sei 2^n die kleinste Potenz von 2, die größer als N ist. Mit anderen Worten, n ist die Input-Länge

$$n = 1 + \lfloor \log_2 N \rfloor .$$

Wir wenden nun den Entscheidungsalgorithmus für $k = 2^{n-1} - 1$ an. Ist die Antwort `NEIN`, dann ist N eine Primzahl, denn jeder nichttrivale Faktor M von N erfüllt $M \leq N/2 < 2^{n-1}$. In diesem Fall sind wir fertig.

Nehmen wir an, die Antwort ist `JA`. Dann wiederholen wir den Algorithmus für das Entscheidungsproblem mit $k = 2^{n-2} - 1$. Falls die Antwort `NEIN` ist, dann muß N einen nichttrivalen Faktor der Form

$$M = 1 \cdot 2^{n-2} + b_{n-3}2^{n-3} + \cdots + b_1 2^1 + b_0$$

haben, wobei die b_i die Bits der Binärdarstellung von M sind. Fall die Antwort `JA` ist, dann muß N einen nichttrivalen Faktor der gleichen Form haben mit dem ersten Bit 0 statt 1, i.e.

$$M = b_{n-3}2^{n-3} + \cdots + b_1 2^1 + b_0.$$

Um das nächste Bit b_{n-3} zu finden setzt man entweder $k = 2^{n-2} + 2^{n-3} - 1$ für den Fall wenn das vorherige Ergebnis des Algorithmus die `NEIN` Antwort war, oder $k = 2^{n-3} - 1$ falls das Ergebnis des Algorithmus im vorherigen Schritt die `JA` Antwort war. Falls der Algorithmus jetzt mit `NEIN` antwortet, wissen wir, dass $b_{n-3} = 1$ ist. Ist die Antwort `JA`, setzt man $b_{n-3} = 0$. In dieser Weise fährt man mit dem Entscheidungsalgorithmus fort, um jedes Bit in dem nichttrivialen Faktor von N zu finden. Nach nur n Anwendungen des Algorithmus wird man einen nichttrivialen Faktor von N finden. Auf diese Weise wird der Algorithmus für das Entscheidungsproblem in den Algorithmus des korrespondierenden Suchproblems umgewandelt.

Beispiel:

Gegeben ist ein Algorithmus für das Faktorisierungs Entscheidungsproblem. Wir wenden diesen an, um die Zahl 91 zu faktorisieren.

`FRAGE 1:` Hat 91 einen Faktor zwischen 2 und 63? `ANTWORT:` Ja

`FRAGE 2:` Hat 91 einen Faktor zwischen 2 und 31? `ANTWORT:` Ja

`FRAGE 3:` Hat 91 einen Faktor zwischen 2 und 15? `ANTWORT:` Ja

`FRAGE 4:` Hat 91 einen Faktor zwischen 2 und 7? `ANTWORT:` Ja

`FRAGE 5:` Hat 91 einen Faktor zwischen 2 und 3? `ANTWORT:` Nein

`FRAGE 6:` Hat 91 einen Faktor zwischen 2 und 5? `ANTWORT:` Nein

`FRAGE 7:` Hat 91 einen Faktor zwischen 2 und 6? `ANTWORT:` Nein

Wir folgern, dass 7 ein nichttrivialer Faktor von 91 ist. Diese Methode der binären Suche mit Verwendung eines Algorithmus für das Faktorisierungs Entscheidungsproblem liefert immer den kleinsten nichttrivialen Faktor von N.

21.4.2 $\mathcal{P}$ und $\mathcal{NP}$

Definition:

Ein Entscheidungsproblem P ist in der Komplexitätsklasse $\mathcal{P}$ der **Polynomialzeit Probleme**, wenn ein Polynom $p(n)$ existiert und ein Algorithmus, so dass für eine Instanz von P mit Inputlänge $\leq n$ der Algorithmus die Frage korrekt in einer Zeit $\leq p(n)$ beantwortet.

Äquivalent dazu ist folgende Definition:

Ein Entscheidungsproblem P ist in $\mathcal{P}$, wenn es eine Konstante c und einen Algorithmus gibt, so dass der Algorithmus eine Instanz des Problems P mit Inputlänge $\leq n$ die Frage in der Zeit $O(n^c)$ beantwortet.

Anmerkungen:

1. Der Begriff *Zeit* in obiger Definition bezieht sich auf die Laufzeit eines Computers, auf dem der Algorithmus implementiert ist. Alternativ kann für den Begriff *Zeit* auch die Anzahl von Bit-Operationen gesetzt werden, die benötigt werden, um den Algorithmus auszuführen.

2. Die obige Definition der Komplexitätsklasse $\mathcal{P}$ versucht die Klasse von Problemen zu beschreiben, die in der Praxis schnell lösbar sind. Es ist *a priori* nicht klar, dass $\mathcal{P}$ die geeignte Klasse für diesen Zweck ist. Ein Algorithmus mit Laufzeit n^{100} mit der Inputlänge n ist langsamer als ein Algorithmus mit Laufzeit $e^{0.0001n}$, es sei denn n ist größer als etwa 10 Millionen. Dennoch ist der erste Algorithmus ein Polynomialzeit Algorithmus, der zweite ein Algorithmus mit exponentieller Laufzeit.

   Die Erfahrung hat jedoch gezeigt, wenn ein Problem von praktischen Interesse in der Komplexitätsklasse $\mathcal{P}$ liegt, dann gibt es einen Algorithmus mit einer Laufzeit, die durch eine *kleine* Potenz der Inputlänge beschränkt ist.

3. Manchmal hat ein Problem, das in $\mathcal{P}$ liegt – oder von dem man annimmt, dass es in $\mathcal{P}$ liegt – einen praktischen, effektiven Algorithmus, der nicht polynomial in der Zeit ist. Ein Beispiel ist das folgende Problem der Primalität:

   `INPUT`: Eine positive ganze Zahl N.

   `FRAGE`: Ist N eine Primzahl?

   Falls die sogenannte *erweiterte Riemannsche Hypothese* wahr ist, dann beantwortet ein Algorithmus von Miller diese Frage in polynomialer Zeit. iWelbst wenn man die erweiterte Riemannsche Hypothese annimmt, hat für $N < 10^{1000}$ der effektivste deterministische Algorithmus, der bekannt ist, eine Laufzeit $n^{O(\ln \ln n)}$, wobei $n = O(\ln N)$ die Inputlänge ist.

Ein Beispiel einer leicht abgewandelten Art ist folgendes Entscheidungsproblem:

`INPUT`: Eine elliptische Kurve E modulo p und eine ganze Zahl k.

`FRAGE`: Gibt es $\geq k$ Punkte auf E?

Ein von SCHOOF[4] entwickelter Algorithmus beantwortet diese Frage in der Laufzeit $O(n^8)$, wobei $n = O)\ln p)$ die Inputlänge ist. Es gibt einen Algorithmus von ATKON, der in der Praxis wesentlich effektiver ist, aber bisher gibt es keine rigorose Ableitung einer oberen Schranke für die Laufzeit, insbesondere ist nicht klar, ob ATKINS Algorithmus in $\mathcal{P}$ liegt.

Definition:

Ein Entscheidungsproblem P ist in der Komplexitätsklasse $\mathcal{NP}$, wenn – gegeben eine Instanz von P – jemand mit unbeschränkter Rechenleistung die Frage beantworten kann, sondern im Fall der Antwort `JA` auch den Beweis liefern kann, den wiederum eine andere Person verwenden kann, um in polynomialer Laufzeit die Korrektheit der Antwort zu prüfen. Der Beweis, dass die `JA` Antwort korrekt ist, nennt man ein **Zertifikat** oder **Polynomialzeit Zertifikat**.

Ein Entscheidungsproblem P liegt in der Komplexitätsklasse $co - \mathcal{NP}$, wenn sie obige Bedingung gilt, die Antwort `JA` durch `NEIN` ersetzt wird. Das bedeutet also, für jede Instanz mit einer `NEIN` Antwort existiert ein Polynomialzeit Zertifikat dass die Antwort `NEIN` korrekt ist.

Beispiel:

Wir betrachten nochmals das Entscheidungsproblem der Faktorisierung:

`INPUT`: Positive ganze Zahlen N und k.

`FRAGE`: Hat N einen Faktor im Intervall $[2, k]$?

Dieses Problem ist sehr wahrscheinlich nicht in $\mathcal{P}$, jedoch sicherlich in $\mathcal{NP}$. Angenommen es gibt ein Orakel, das die Zahl N faktorisieren kann und erhält das Resultat, dass N einen Faktor $M \in [2, k]$ hat. Nachdem die Antwort `JA` gegeben wird, wird auch die Zahl M geliefert. Dann kann mit polynomialem Zeitaufwand die Korrektheit der Antwort verifiziert werden, indem man einfach die Zahl N durch M dividiert.

Das Faktorisierungsproblem ist auch in der Klasse $co - \mathcal{NP}$. Ist die Antwort zur obigen Frage `NEIN`, dann liefert das Orakel die komplette Faktorisierung von N. Daraus ist unmittelbar ersichtlich, dass es keinen Faktor im Intervall $[2, k]$ gibt. Gleichzeitig muß das Orakel auch ein Zertifikat liefern mit dessen Hilfe man in polynomialem Zeitaufwand verifizieren kann, dass jeder Faktor auch eine Primzahl ist[5]. Es gibt eine Reihe von Zertifikaten.

[4] Siehe R. SCHOOF (1985) *Elliptic curves over finite fields and the computation of square roots mod p*, Math Comp. **44**, 483 – 494.

[5] Ohne dieses Zertifikat könnte einer der Faktoren weiter faktorisiert werden und einen Faktor im Intervall $[2, k]$ liefern.

Beispiel:

Das Traveling Salesman Problem ist mit großer Wahrscheinlichkeit nicht in $\mathcal{P}$. Es liegt jedoch in $\mathcal{NP}$. Das bedeutet, angenommen das Orakel findet eine Tour für den Handlungsreisenden mit Länge kleiner oder gleich k. Das Orakel liefert die Antwort `JA` und stellt die Lösung zur Verfügung. Es ist dann mit polynomialem Aufwand möglich, die Korrektheit der Antwort zu verifizieren[6].

Auf die gleiche Weise läßt sih zeigen, dass das 3-Farben Problem in $\mathcal{NP}$ liegt.

Falls ein Problem in $\mathcal{P}$ liegt, ist es trivialerweise auch in $\mathcal{NP}$. Das heißt also

$$\mathcal{P} \subset \mathcal{NP}.$$

Es ist fast sicher, dass die Komplexitätsklasse $\mathcal{NP}$ eine viel größere Klasse von Problemen ist als die Klasse $\mathcal{P}$, dies ist aber nicht beweisen. Die Behauptung $\mathcal{P} \neq \mathcal{NP}$ ist die berühmteste Vermutung der Computerwissenschaft.

21.4.3 Reduktion eines Problems auf ein anderes

Definition:

Seien P_1 und P_2 zwei Entscheidungsprobleme. Man sagt, das Problem P_1 reduziert sich auf das Problem P_2, wenn es einen Polynomialzeit Algorithmus gibt als Funktion der Inputlänge von P_1. Gegeben ist eine Instanz p_1 des Problems P_1, dann konstruiert dieser Algorithmus eine Instanz p_2 von P_2, so dass die Antwort für P_1 die gleiche ist wie die Antwort für P_2.

Eine grundlegende Anwendung des Konzeptes der Reduktion ist die folgende. Angenommen wir haben einen effektiven Algorithmus für ein Entscheidungsproblem P_2.Falls sich das Problem P_1 auf das Problem P_2 reduzieren läßt, können wir den Algorithmus zur Lösung von P_2 auch zur Lösung von P_1 benutzen. Ist nämlich eine Instanz von P_1 gegeben, dann können wir in polynomialer Laufzeit eine korrespondierende Instanz des Problems P_2 finden, indem wir den in obiger Definition erwähnten Algorithmus anwenden. Wenden wir dann den Algorithmus für das Problem P_2 auf diese Instanz von P_2 an, ist die erhaltene Antwort auch eine Antwort für das ursprüngliche Entscheidungsproblem P_1. Das bedeutet, ein Algorithmus für P_2 liefert automatisch auch einen Algorithmus für P_1. Ist der Algorithmus für das Entscheidungsproblem P_2 ein Polynomialzeit Algorithmus, dann ist auch P_1 in polynomialer Laufzeit lösbar.

Beispiel:

Sei P_1 das folgende Entscheidungsproblem:

[6] Zu verifizieren ist lediglich, dass der Weg kleiner als die gegebene Zahl k ist, es ist *nicht* zu verfizieren, dass der gefundene Weg der kürzeste Weg überhaupt ist.

`INPUT`: Ein quadratisches Polynom $p(X)$ mit ganzzahligen Koeffizienten.

`FRAGE`: Hat $p(X)$ zwei verschiedene reelle Nullstellen?

Sei P_2 das Entscheidungsproblem:

`INPUT`: Eine ganze Zahl N.

`FRAGE`: Ist N positiv?

Wir zeigen, dass sich das Problem P_1 auf das Problem P_2 reduzieren läßt. Sei

$$p(X) = aX^2 + bX + c$$

eine Instanz des Problems P_1. Sei

$$N = b^2 - 4ac$$

was in polynomialer Luafzeit berechnet werden kann. Dann hat das Problem P_2 mit Input N_1 eine `JA` Antwort genau dann, wenn das Entscheidungsproblem P_1 mit dem Input $p(X) = ax^2 + bX + c$ eine `JA` Antwort hat.

Die obige Definiton kann auch in der umgekehrten Weise eingesetzt werden. Angenommen wir wissen – oder vermuten – dass ein Problem P_1 sehr schwierig ist. Das bedeutet, wir sind davon überzeugt, dass es keinen effektiven Algorithmus für das Problem P_1 gibt. Falls sich das Problem P_1 auf das Problem P_2 reduziert, dann folgt, dass es für P_2 ebenfalls keinen effektiven Algorithmus gibt.

Die obige Definition ist etwas zu restriktiv. Es lohnt sich, eine breiter gefaßte Definition der polynomialen Zeitreduktion von P_1 auf P_2 zu betrachten. Diese erlaubt uns, mehrere verschiedene Instanzen des Problems P_2 zu verwenden, um eine Instanz des Problems P_1 zu lösen. Wir benötigen zunächst den begriff eines *Orakels*.

Definition:

Sei P_2 ein Entscheidungs- oder ein Suchproblem. Sagen wir bei der beschreibung eines Algorithmus eines anderen Problems P_1 *Aufruf eines P_2-Orakels*, dann bedeutet das, dass der Algorithmus eine Instanz von P_2 generiert hat und wir nehmen weiterhin an, dass ein anderer Algorithmus uns die Antwort des entsprechenden P_2 Outputs gibt. Die Zeit, die der Algorithmus für P_2 benötigt wird in der Laufzeit des Algorithmus für das Problem P_1 nicht berücksichtigt. Mit anderen Worten, wir nehmen an, der Algorithmus für P_2 ist eine Black-Box, die ohne Zeitaufwand wirkt.

Im Sprachgebrauch der Informatik kann ein Orakel als Unterprogramm aufgefaßt werden, das jederzeit aufgerufen werden kann und dessen Laufzeit nicht in der Gesamtlaufzeit des Programms aufgeführt wird.

Definition:

Seien P_1 und P_2 zwei Probleme – entweder beides Entscheidungs- oder beides Suchprobleme. Wir sagen, das Problem P_1 reduziert sich auf das Problem P_2 in polynomialer Zeit, wenn es einen Polynomialzeit Algorithmus für P_1 gibt, der höchstens polynomial viele Aufrufe an eine P_2-Orakel benötigt.

Beispiel:

Sei P_1 das Faktorisierungs Suchproblem und sei P_2 das Faktorisierungs Entscheidungsproblem. Wir haben gesehen, dass P_1 mit n Aufrufen eines P_2-Orakels gelöst werden kann, wobei n die Inputlänge ist. Daher reduziert sich P_1 auf das Problem P_2.

21.4.4 $\mathcal{NP}$-vollständige Probleme

Definition:

Ein Entscheidungsproblem P in $\mathcal{NP}$ heißt $\mathcal{NP}$-vollständig, wenn wenn jedes andere Problem Q in $\mathcal{NP}$ auf P in polynomialer Zeit reduziert werden kann.

Diese Definition bedeutet also: Hätte man einen Polynomialzeit-Algorithmus für ein $\mathcal{NP}$-vollständiges Problem P, dann hätte man auch Polynomialzeit-Algorithmen für alle anderen $\mathcal{NP}$ Probleme Q. Das impliziert

$$\mathcal{P} = \mathcal{NP}$$

und die $\mathcal{P} \neq \mathcal{NP}$ Vermutung wäre falsch. Aus diesem Grund ist es sehr unwahrscheinlich, dass irgendjemand einen Polynomialzeit Algorithmus für ein $\mathcal{NP}$-vollständiges Problem präsentieren wird. In einem bestimmten sinne sind die $\mathcal{NP}$-vollständigen Probleme die schwierigsten Probleme in der Komplexitätsklasse $\mathcal{NP}$. Siehe dazu auch [91, Chap. 1.5] oder [165].

Diese Aussage muß mit Vorsicht angesehen werden. Es kann durchaus sein, dass man in der Lage ist, einne effektiven Algorithmus – unter Umständen sogar einen Polynomialzeit Algorithmus – für die meisten Instanzen eines bestimmten $\mathcal{NP}$-vollständigen Problems anzugeben. Dies widerspricht *nicht* der $\mathcal{P} \neq \mathcal{NP}$ Vermutung.

In der Praxis ist es üblicherweise sehr schwierig, große Instanzen eines $\mathcal{NP}$-vollständigen Problems effektiv zu lösen. Oftmals wächst die Laufzeit aller bekannter Algorithmen exponentiell mit der Länge des Inputs.

Man kann zeigen (siehe e.g. Garey and Johnson [91], pg. 95 –96), dass das Traveling Salesman Problem $\mathcal{NP}$-vollständig ist. Angenommen man hat eine sehr komplizierte Instanz des TSP mit mehreren hundert Städten. Es kann

Millionen von Millionen Jahren dauern bis selbst die schnellsten Rechner einen optimalen Weg dieser instanz des TSP gefunden haben[7].

Man kann zeigen, dass das 3-Farben Problem ebenfalls $\mathcal{NP}$-vollständig ist.

Schließlich ist noch anzumerken, dass es möglich ist, dass sich ein Problem P auf ein $\mathcal{NP}$ Problem reduzieren läßt, obwohl P selbst wahrscheinlich nicht in $\mathcal{NP}$ liegt.

Beispiel:

Das **exakte Traveling Salesman Problem** ist das folgende Entscheidungsproblem:

> `INPUT`: Eine Menge von Städten und Abstände zwischen diesen Städten, eine ganze Zahl k.
>
> `FRAGE`: Gibt es eine kürzeste Tour durch alle Städte mit Länge exakt gleich k?

Um ein Zertifikat für eine `JA` Antwort zu liefern, ist es nicht ausreichend, einfach eine Tour der Länge k anzugeben. Das Orakel muß (irgendwie) uns davon überzeugen, dass es keine kürzere Tour gibt. Es ist sehr unwahrscheinlich, dass es solch ein Zertifikat gibt. Andererseits kann man die binäre Suchmethode anwenden, um das exakte Traveling Salesman Entscheidungsproblem auf das Traveling Salesman Entscheidungsproblem zu reduzieren. Es ist auch möglich die Umkehrung zu zeigen, i.e. die Reduktion des Traveling Salesman Entscheidungsproblem auf das exakte Traveling Salesman Entscheidungsproblem [165, pp. 411-412]. In diesem Fall sagt man, dass zwei Probleme polynomial Zeitäquivalent sind, manchmal bezeichnet man dies auch als $\mathcal{NP}$-äquivalent.

Wir beenden diesen Abschnitt mit einer definition, die auf Probleme angewendet werden, die nicht notwendigerweise in $\mathcal{NP}$ liegen.

Definition:

> Ein Entscheidungs- oder Suchproblem heißt **$\mathcal{NP}$-hart**, wenn irgendein $\mathcal{NP}$-Problem sich auf diese Problem reduzieren läßt.

[7] Eine erschöpfende Diskussion des Traveling Salesman Problems findet man in [3].

Kapitel 22

Galois Felder

Wie wir im Abschnitt [15.5] gezeigt haben, ist das Tripel $(\mathbb{Z}_p, +, \cdot)$ genau dann ein Körper, wenn p eine Primzahl ist. Die Körpereigenschaften sind für viele moderne Kryptosysteme essentiell, sobald die Multiplikationsoperation eine Rolle spielt. Der Grund liegt darin, dass die Dechiffrierung eines Verschlüsselungsschemas mit Multiplikation das multiplikative Inverse erfordert. Das wiederum existiert genau dann, wenn die zugrundeliegende Menge von Zahlen mit der Addition und Multiplikation als binäre Operation einen Körper bilden. Wie in Kapitel [5] dargestellt ist, verwendet der im Advanced Encryption Standard AES eingesetzte Rijndael Algorithmus Operationen auf Bytes. Da $2^8 = 256$ keine Primzahl ist, ist es zwecklos, auf $(\mathbb{Z}_{2^8}, +, \cdot)$ zu arbeiten.

Die erforderliche mathematische Struktur, um mit Mengen arbeiten zu können, deren Kardinalität kein Primzahl ist und die dennoch die Eigenschaften eines Körpers haben, ist die sogenannte **Körpererweiterung**. Durch dieses Konstrukt ist es möglich, einen endlichen Körper $\texttt{GF}(2^8)$ mit 256 Elementen zu untersuchen. Die Arithmetik auf diesem Körper ist jedoch definiert durch die Addition und Multiplikation von **Polynomen**. Daher müssen wir uns in diesem Kapitel ausgiebig mit Polynomen und deren Eigenschaften befassen. In der Mathematik nennt man diese Art von Operationen **Arithmetik modulo irreduzibler Polynome vom Grad** m, deren Koeffizienten ganze Zahlen modulo p sind. p ist dabei eine Primzahl. Diese Körper werden mit $\texttt{GF}(p^m)$ oder $\mathbb{F}_{p^m}$ bezeichnet. Die komplette Arithmetik wird modulo $p(x)$ durchgeführt, wobei $p(x)$ ein **irreduzibles Polynom vom Grade** m **ist.**

Wir haben bereits endlichen Zahlenfelder der Ordnung p kennengelernt. Diese Körper bezeichnen wir mit dem Symbol $\texttt{GF}(p)$, wobei GF ein Akronym zu Ehren von Evariste Galois ist (siehe Abbildung [22.1]). Solche endlichen Körper — auch Galois Felder genannt — bestehen beispielsweise aus den Elementen $0, 1, \ldots, p-1$, für die die grundlegenden arithmetischen Operationen wie Addition, Multiplikation, Subtraktion und Division (ohne 0) wohldefiniert sind. Diese binären Operationen erfüllen die geforderten Kommutativ–, Assoziativ– und Distributivgesetze.

So hat das Galois Feld $\texttt{GF}(3) = \mathbb{F}_3$ die drei Elemente 0,1,2 mit den in der Tabelle [22.1] gezeigten Verknüpfungen.

Abbildung 22.1: Evariste Galois (1811 – 1832)

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Tabelle 22.1: Addition und Multiplikation im Körper GF(3)

Der nächste Schritt besteht in der Konstruktion endlicher Zahlenkörper, deren Ordnung eine Potenz einer Primzahl ist, p^m. Diese Körper bezeichnen wir mit $\mathtt{GF}(p^m)$. Diese Galois Felder haben sehr viele Anwendungen in der Kryptologie und in anderen Disziplinen wie Physik oder Codierungstheorie. Das Galois Feld $\mathtt{GF}(p^m)$ kann auf unterschiedliche Weise realisiert werden:

- ↪ als m–Tuples, *i.e.* Vektoren,
- ↪ als $m \times m$ Matrizen,
- ↪ als Polynome vom Grad $m - 1$.

Alle Konzepte arbeiten dabei mit Komponenten bzw. Koeffizienten aus dem Körper $\mathtt{GF}(p)$. Diese drei Darstellungen des Körpers $\mathtt{GF}(p^m)$ sind alle zueinander isomorph.

Einführungen in diesen Themenkreis findet man in [208], [20], [212], [152]. Die Standard–Referenz zu diesem Themenkreis ist die Monographie von Lidl und Niederreiter [144].

22.1 Endliche Körper der Form GF(p)

Im Abschnitt [13.4] haben wir die algebraische Struktur eines Körper definiert als eine Menge mit zwei binären Verknüpfungen, die die Axiome in der Definition [??] erfüllen. Körper mit unendlich vielen Elementen — beispielweise die Menge $\mathbb{Q}$ mit der Addition und Multiplikation von rationalen Zahlen oder die reellen Zahlen mit Addition und Multiplikation — sind nicht zweckmäßig für kryptographische Anwendungen. Endliche Körper spielen jedoch eine wesentliche Rolle in vielen kryptographischen Algorithmen. Man kann zeigen, dass die **Ordnung** eines endlichen Körpers[1] eine Potenz einer Primzahl p^n ist, wobei n eine natürliche Zahl ist.

Der endliche Körper der Ordnung p^n wird mit GF(p^n) bezeichnet, wobei GF für Galois Field steht. Für unsree Zwecke sind zwei spezielle Fälle von Interesse, das ist der Fall $n = 1$, wobei wir endliche Körper GF(p) betrachten, die eine andere Struktur haben als endliche Körper mit $n > 1$. Im Abschnitt [??] sehen wir uns endliche Körper der Form GF(2^n) an.

22.1.1 Endliche Körper der Ordnung p

Für eine vorgegebene Primzahl p definieren wir den endlichen Körper der Ordnung p, GF(p) als die Menge

$$\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$$

zusammen mit den arithmetsichen Operationen modulo p. Wie haben bereits gesehen, dass die Menge $\mathbb{Z}_n$ der ganzen Zahlen $\{0, 1, 2, \ldots, n-1\}$ zusammen mit den arithmetischen Operationen modulo n die algebraische struktur eines kommutativen Rings bilden. Wir haben weiterhin bereits gesehen, dass eine Zahl in $\mathbb{Z}_n$ ein multiplikatives Inverses genau dann hat, wenn diese Zahl coprim zu n ist. Falls die Zahl n eine Primzahl ist, dann sind alle nichtverschwindenden Zahlen in $\mathbb{Z}_n$ relativ prim zu n, daher existiert für jede Zahl in $\mathbb{Z}_n \setminus \{0\}$ ein multiplikatives Inverses . Daher ist:

$$\text{Für alle } w \in \mathbb{Z}_p,\ w \neq 0 \text{ existiert ein } z \in \mathbb{Z}_p, \text{ so dass } w \cdot z \equiv 1 \bmod p.$$

Wenn man alle Element von $\mathbb{Z}_p$ mit w multipliziert — weil w coprim zu p ist — sind die durch diese Operation entstehenden Restklassen wieder alle Elemente von $\mathbb{Z}_p$, jedoch gegenüber der ursprünglichen Anordnung permutiert. Daher hat genau eine der Restklassen den Wert 1. Daher existiert eine ganze Zahl in $\mathbb{Z}_p$, die mit w multipliziert den Wert 1 ergibt. Diese ganze Zahl ist das multiplikative Inverse von w, welches wir mit w^{-1} bezeichnen. Daher ist $\mathbb{Z}_p$ ein endlicher Körper der Ordnung p.

In $\mathbb{Z}_p$ gilt dann die Aussage

$$\text{Wenn } a \cdot c \equiv (a \cdot b) \bmod p, \text{ dann ist } c \equiv b \bmod p. \tag{22.1}$$

Beispiel [22.51]

[1] Darunter versteht man die Anzahl der Elemente der zugrunde liegenden Menge.

Wir betrachten die Menge

$$\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\}$$

mit den beiden Operationen

$$a + b \bmod 7$$

und

$$a \cdot b \bmod 7 \qquad \text{für alle } a, b \in \mathbb{Z}_7.$$

Diese Verknüpfungen können wir in Form von Tabellen darstellen.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Tabelle 22.2: Addition modulo 7.

Die Multiplikation modulo 7 liefert die Tabelle [22.3].

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Tabelle 22.3: Multiplikation modulo 7.

Die Tabelle [22.4] listet die additiven und multiplikativen Inversen modulo 7 auf.

22.1.2 Berechnen des multiplikativen Inversen in $\mathtt{GF}(p)$

Für kleine Werte von p ist es einfach, das multiplikative Inverse eines Elements in $\mathtt{GF}(p)$ zu finden. Man erstellt einfach eine Multiplikationstabelle wie die Tabelle

w	$-w$	w^{-1}
0	0	$--$
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

Tabelle 22.4: Additive und multiplikative Inverse modulo 7.

[22.3] für den Fall $GF(7)$ und liest das gesuchte Ergebnis einfach ab. Für große Werte von p ist dies natürlich kein praktikabler Zugang.

Falls die Zahlen a und b relativ prim sind, dann hat b ein multiplikatives Inverses modulo a. Wenn also $\mathtt{ggT}(a,b) = 1$ gilt dann hat b ein multiplikatives Inverses modulo a. Das heisst wiederumg für positive ganze Zahlen $b < a$ existiert ein b^{-1}, so dass

$$b \cdot b^{-1} \equiv 1 \bmod a.$$

Wenn a eine Primzahl ist, dann sind a und b offensichtlich relativ prim und haben damit den größten gemeinsamen Teiler 1. Mit Hilfe des erweiterten EUKLIDischen Algorithmus kann das multiplikative Inverse b^{-1} systematisch berechnet werden.[2]

[2] Siehe dazu Kapitel [18.2].

22.2 Arithmetik mit Polynomen

Bevor wir mit der Untersuchung von endlichen Körpern fortfahren, ist es notwendig, sich mit der Arithmetik von Polynomen zu befassen. Wir betrachten Polynome mit einer Variablen x und unterscheiden drei Fälle:

- Gewöhnliche Arithmetik von Polynomen, die die algebraischen Gesetze verwenden.
- Arithmetik von Polynomen bei denen die Arithmetik der Koeffizienten modulo p ausgeführt wird, *i.e.* die Koeffizienten der Polynome sind Elemente von $\mathtt{GF}(p)$.
- Arithmetik von Polynomen bei denen die Koeffizienten Elemente von $\mathtt{GF}(p)$ sind und die Polynome sind definiert modulo einem Polynom $m(x)$, dessen höchste Potenz eine natürliche Zahl n ist.

In diesem Kapitel betrachten wir die ersten beiden Fälle, der dritte Fall wird in Abschnitt [??] untersucht.

22.2.1 Gewöhnliche Arithmetik mit Polynomen

In der elementaren Algebra werden Polynome als Ausdrücke der Form[3]

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

betrachtet. Die Größen a_i nennt man *Koeffizienten* und sind üblicherweise reelle oder komplexe Zahlen. Die Größe x wird üblicherweise als Variable betrachtet, *i.e.* substituiert man eine beliebige Zahl b für die Variable x, dann erhält man ein Element des zugrundeliegenden Körpers, in der elementaren Algebra ist dann üblicherweise $f(b) \in \mathbb{R}$, *i.e.* es ergibt sich eine reelle Zahl. Dieses Konzept der Polynome über dem Körper der reellen Zahlen kann auf beliebige Körper verallgemeinert werden.

Im Kontext der abstrakten Algebra ist man nicht an der Auswertung eines Polynomes an einem bestimmten Wert für x interessiert.[4] Um diesen Sachverhalt zu betonen, bezeichnet man x als **Unbestimmte**.

Die Arithmetik von Polynomen umfasst zunächst die Addition, Subtraktion und Multiplikation. Division betrachten wir spätrer. Diese Operationen sind kanonisch definiert, man behandelt die Unbestimmte x wie ein Element des zugrunde liegenden Körpers (*e.g.* $\mathbb{R}$ oder $\mathbb{C}$).

Die beiden Polynome

$$f(x) = \sum_{i=0}^{n} a_i x^i \quad \text{und} \quad g(x) = \sum_{i=0}^{n} b_i x^i$$

[3] In diesem Abschnitt folgen wir der Darstellung in [144] und [208].
[4] Beispielsweise die Berechnung von $f(12)$.

über R sind gleich, genau dann, wenn $a_i = b_i$ für alle $0 \leq i \leq n$. Die *Summe* zweier Polynome und die Differenz ist definiert durch

$$f(x) \pm g(x) = \sum_{i=0}^{n} (a_i \pm b_i) x^i.$$

Um das *Produkt* zweier Polynome zu definieren, setzen wir:

$$f(x) = \sum_{i=0}^{n} a_i x^i \quad \text{und} \quad g(x) = \sum_{j=0}^{m} b_j x^j$$

Dann ist:

$$f(x) \cdot g(x) = \sum_{k=0}^{n+m} c_k x^k$$

wobei die Koeffizienten des Produktes gegeben sind durch:

$$c_k = \sum_{i+j=k} a_i \cdot b_j \quad 0 \leq i \leq n \; ; \; 0 \leq j \leq m$$

Beispiel [22.52]

Sei

$$f(x) = x^4 + 2x^3 + x + 5$$

und

$$g(x) = x^3 - 2x + \frac{1}{2}.$$

Hier ist die zugrundeliegende Menge für die Koeffizienten die Menge der rationalen Zahlen $\mathbb{Q}$. Dann ist

$$\begin{aligned} f(x) + g(x) &= x^4 + 3x^3 + 3x + \frac{11}{2}, \\ f(x) - g(x) &= x^4 + x^3 - x + \frac{9}{2}, \\ f(x) \times g(x) &= x^7 + 2x^6 - 2x^5 - \frac{5}{2}x^4 + 6x^3 - 2x^2 - \frac{19}{2}x + \frac{5}{2}. \end{aligned}$$

Beispiel [22.53]

Sei $R = (\mathbb{Z}, +\cdot)$ der Ring der ganzen Zahlen mit der üblichen Addition und Multiplikation. Wir setzen:

$$\begin{aligned} f(x) &= 4x^3 + 7x^2 + 12x, \\ g(x) &= 3x^4 + 12x^2 + x + 1. \end{aligned}$$

Damit ist also:

$$\begin{aligned} f(x) &= \sum_{i=0}^{n} a_i x^i \\ &= a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0 x^0 \\ &= 4x^3 + 7x^2 + 12x, \end{aligned}$$

daher sind die Koeffizienten des ersten Polynoms $f(x)$:

$$a_3 = 4; a_2 = 7; a_1 = 12; a_0 = 0.$$

Analog erhält man für das zweite Polynom $g(x)$ die Koeffizienten:

$$b_4 = 3; b_3 = 0; b_2 = 12; b_1 = 1; b_0 = 1.$$

Damit sind die Koeffizienten des Produktpolynoms gemäß:

$$c_k = \sum_{i+j=k} a_i \cdot b_j \quad 0 \leq i \leq n \; ; \; 0 \leq j \leq m$$

gegeben durch:

$$\begin{aligned} c_0 &= \sum_{i+j=0} a_i \cdot b_j \\ &= a_0 \cdot b_0 = 0 \\ c_1 &= \sum_{i+j=1} a_i \cdot b_j \\ &= a_0 \cdot b_1 + a_1 \cdot b_0 = 12 \\ c_2 &= \sum_{i+j=2} a_i \cdot b_j \\ &= a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 = 19 \\ c_3 &= \sum_{i+j=3} a_i \cdot b_j \\ &= a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0 = 155 \\ c_4 &= \sum_{i+j=4} a_i \cdot b_j \\ &= a_0 \cdot b_4 + a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1 + a_4 \cdot b_0 = 88 \\ c_5 &= 84 \\ c_6 &= 21 \\ c_7 &= 12 \end{aligned}$$

Damit ist:

$$\begin{aligned} f(x) \cdot g(x) &= \sum_{k=0}^{n+m} c_k x^k \\ &= 12x^7 + 21x^6 + 84x^5 + 88x^4 + 155x^3 + 19x^2 + 12x. \end{aligned}$$

Wie diese Beispiele zeigen, können wir also Polynome addieren, subtrahieren und multiplizieren, es ist nicht schwierig zu zeigen, dass die Menge der Polynome über einer Koeffizientenmenge R die algebraische Strukur eines kommutativen Rings hat.

Definition [22.21]:

Der Ring, der von den Polynomen über der Koeffizientenmenge R mit den oben definierten Additions– und Multiplikationsoperationen erzeugt wird, heißt **Polynomring** über R und wird mit $R[x]$ bezeichnet.

Das neutrale Element der Addition des Polynomrings $R[x]$ ist das Polynom, dessen sämtliche Koeffizienten den Wert 0 haben.

Definition [22.22]:

Sei

$$f(x) = \sum_{k=0}^{n} a_k x^k = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \cdots + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

ein Polynom über einem Körper F, welches nicht das Nullpolynom ist. Daher können wir annehmen, dass $a_n \neq 0$. Dann heißt a_n der *führende Koeffizient* von f und a_0 ist der *konstante Term*. n heißt der *Grad des Polynoms* $f(x)$, symbolisch:

$$n = \deg\,(f(x)) = \deg\,(f)$$

Per Konvention setzt man $\deg(0) = -\infty$. Ist der führende Koeffizient von $f(x)$ gleich 1, dann heißt $f(x)$ *normiertes Polynom*.[5]

22.2.2 Polynom–Arithmetik mit Koeffizienten in $\mathbb{Z}_p$

Wir betrachten nun Polynome, deren Koeffizienten Elemente des Körpers $\mathtt{GF}(p)$ sind. Man spricht dann von **Polynomen über einem Körper K.** Auch in diesem Fall bilden die Polynome die algebraische Struktur eines Rings, man nennt diesen Ring den **Polynomring**. Man fasst dabei jedes einzelne Polynom auf als Element dieses Rings.

Betrachtet man die Arithmetik von Polynomen über einem Körper, dann ist die Division möglich. Man beachte, dass dies nicht heisst, dass eine *exakte Division* möglich ist. In einem Körper bedeutet das: Sind zwei Elemente a und b dieses Körpers gegeben, dann ist der Quotient a/b ebenfalls ein Element des Körpers. Betrachtet man jedoch einen Ring R, der nicht die Körperstruktur hat, dann ergibt die Division einen Quotienten und einen Rest. Dies bezeichnen wir mit *keine* exakte Division.

Beispiel [22.54]

Wir betrachten die Division $5/3$ in einer Menge M. Ist $M = \mathbb{Q}$, die Menge der rationalen Zahlen, dann ist das Ergebnis einfach $5/3$, was auch eine rationale Zahl ist und daher ein Element von $\mathbb{Q}$ ist.

Betrachte nun den Fall, dass $M = \mathbb{Z}_7$ ist. In diesem Fall ist

$$\frac{5}{3} = \left(5 \times 3^{-1}\right) \bmod 7 = (5 \times 5) \bmod 7 = 4,$$

was ein exaktes Ergebnis ist, weil $4 \in \mathbb{Z}_7$.

Betrachte schließlich $M = \mathbb{Z}$, die Menge der ganzen Zahlen, die ein Ring bilden, aber keinen Körper. Dann erzeugt $5/3$ einen Quotienten und einen Rest:

$$\frac{5}{3} = 1 + \frac{2}{3},$$

oder

$$5 = 1 \times 3 + 2.$$

Daher ist die Division nicht exakt auf der Menge der ganzen Zahlen.

Betrachten wir nun die Polynomdivision über einer Koeffizientenmenge, die nicht die Struktur eines Körpers hat, dann findet man, dass die Division nicht immer definiert ist.

Beispiel [22.55]

Ist die Koeffizientenmenge M gleich der Menge der ganzen Zahlen, dann hat

$$\frac{5x^2}{3x}$$

keine Lösung, denn diese erfordert einen Koeffizienten mit Werten in $\mathbb{Z}$, was für $5/3$ nicht zutrifft. Führt man die gleiche Operation auf der Koeffizientenmenge $\mathbb{Z}_7$ durch, dann ist

$$\frac{5x^2}{3x} = 4x,$$

dies ist ein wohldefiniertes Polynom über der Menge $\mathbb{Z}_7$.

Wie wir sehen werden ist die Polynomdivision selbst dann nicht notwendigerweise exakt, wenn die Koeffizientenmenge ein Körper ist. Im allgemeinen liefert die Polynomdivision einen Quotienten und einen Rest. Wir können den Divisionsalgorithmus für Polynome über einem Körper folgendermaßen formulieren.

Gegeben ist ein Polynom $f(x)$ vom Grad n und ein zweites Polynom $g(x)$ vom Grad m mit $n \geq m$. Dividieren wir $f(x)$ durch $g(x)$, dann erhalten wir einen Quotienten $q(x)$ und einen Rest $r(x)$, und es gilt

$$f(x) = q(x) \cdot g(x) + r(x). \tag{22.2}$$

Dabei sind die Grade:

$$\begin{aligned} \deg[f(x)] &= n, \\ \deg[g(x)] &= m, \\ \deg[q(x)] &= n - m, \\ \deg[r(x)] &\leq m - 1. \end{aligned}$$

Lassen wir bei der Polyonomdivision Reste im obigen Sinn zu, dann können wir sagen, dass die Division möglich ist, falls die Koeffizientenmengen die algebraische Stuktur eines Körpers hat.

In Analogie zur Integer–Arithmetik können wir für den Rest der Division in Gl. (22.2) schreiben

$$r(x) \equiv f(x) \bmod g(x). \tag{22.3}$$

Wenn bei der Polynomdivision kein Rest entsteht, *i.e.* $r(x) = 0$, dann sagen wir, dass $g(x)$ das Polynom $f(x)$ teilt, formal

$$g(x) \mid f(x).$$

Äquivalent dazu ist die Aussage, dass $g(x)$ ein Faktor von $f(x)$ ist, oder $g(x)$ ist Teiler von $f(x)$.

Beispiel [22.56]

Seien die beiden Polynome

$$f(x) = x^3 + x^2 + 2$$

und

$$g(x) = x^2 - x + 1$$

gegeben, die Koeffizientenmenge ist $\mathbb{R}$. Die Polynomdivision ergibt

$$
\begin{array}{l}
(x^3+x^2+2):(x^2-x+1)=x+2 \\
\underline{-(x^3-x^2+x)} \\
\quad 2x^2-x+2 \\
\quad \underline{-(2x^2-2x+2)} \\
\quad\quad x
\end{array}
$$

Daher ist der Quotient

$$q(x) = x + 2$$

und der Rest

$$r(x) = x.$$

Man verifiziert leicht, dass gilt:

$$\begin{aligned} q(x) \cdot g(x) + r(x) &= (x+2) \cdot (x^2 - x + 1) + x \\ &= (x^3 + x^2 - x + 2) + x \\ &= x^3 + x^2 + 2) \\ &= f(x). \end{aligned}$$

Für unsere Zwecke sind Polynome über dem Körper $\mathtt{GF}(2)$ von besonderem Interesse. Wie wir wisssen ist die Addition auf $\mathtt{GF}(2)$ äquivalent zur XOR–Operation und Multiplikation zur AND–Operation. Weiterhin sind auf $\mathtt{GF}(2)$ die Addition und Subtraktion gleich, da

$$\begin{aligned} 1 + 1 &= 0, \\ 1 - 1 &= 0, \\ 1 + 0 &= 1, \\ 1 - 0 &= 1, \\ 0 + 1 &= 1, \\ 0 - 1 &= 1. \end{aligned}$$

Beispiel [22.57]

Betrachte die beiden Polynome

$$\begin{aligned} f(x) &= x^7 + x^5 + x^4 + x^3 + x + 1, \\ g(x) &= x^3 + x + 1 \end{aligned}$$

über dem Körper GF(2). Dann ist die Addition dieser beiden Polynome

$$\begin{aligned} f(x) + g(x) &= \left(x^7 + x^5 + x^4 + x^3 + x + 1\right) + \left(x^3 + x + 1\right) \\ &= x^7 + x^5 + x^4 + (1+1)x^3 + (1+1)x + 1 + 1 \\ &= x^7 + x^5 + x^4. \end{aligned}$$

Die Subtraktion ergibt:

$$\begin{aligned} f(x) - g(x) &= \left(x^7 + x^5 + x^4 + x^3 + x + 1\right) - \left(x^3 + x + 1\right) \\ &= x^7 + x^5 + x^4 + (1-1)x^3 + (1-1)x + 1 - 1 \\ &= x^7 + x^5 + x^4. \end{aligned}$$

Dies zeigt, dass auf GF(2) die Addition und Subtraktion identisch sind.

Die Multiplikation dieser beiden Polynome ergibt

$$\begin{aligned} f(x) \cdot g(x) &= \left(x^7 + x^5 + x^4 + x^3 + x + 1\right) \cdot \left(x^3 + x + 1\right) \\ &= x^{10} + x^8 + x^7 \\ &\quad + x^8 + x^6 + x^5 \\ &\quad + x^7 + x^5 + x^4 \\ &\quad + x^6 + x^4 + x^3 \\ &\quad + x^4 + x^2 + x \\ &\quad + x^3 + x + 1 \\ &= x^{10} + x^4 + x^2 + 1. \end{aligned}$$

Man beachte bei der Ausf.ührung der Multiplkation, dass sämtliche Produkte modulo 2 zu nehmen sind.

Die Division ergibt sich folgendermaßen, man verwendet das gleiche Verfahren wie bei der Polynomdivision über den reellen Zahlen, hier ist jedoch zu berücksichtigen, dass alle Additionen modulo 2 sind und die Subtrakion ist äquivalent zur Addition (auf GF(2)).

$$\begin{array}{l} (x^7 + x^5 + x^4 + x^3 + x + 1) : (x^3 + x + 1) = x^4 + 1 \\ \underline{x^7 + x^5 + x^4} \\ \qquad\quad x^3 + x + 1 \\ \qquad\quad \underline{x^3 + x + 1} \\ \qquad\quad 0 \end{array}$$

Da die Division den Rest 0 ergibt, ist $g(x)$ ein Teiler von $f(x)$.

Beispiel [22.58]
Sei

$$f(x) = 2x^5 + x^4 + 4x + 3 \in \mathtt{GF}(5)[x]$$

und

$$g(x) = 3x^2 + 1 \in \mathtt{GF}(5)[x].$$

Also, wir arbeiten auf dem Koeffizientenkörper

$$\mathtt{GF}(5) = \mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$$

und $\mathtt{GF}(5)[x]$ bezeichnet die Menge alle Polynome über $\mathtt{GF}(5)$.
Zu zeigen ist, dass das Polynom $f(x)$ geschrieben werden kann in der Form

$$f(x) = q(x) \cdot g(x) + r(x)$$

mit $\deg(r) < \deg q$. Wir verwenden dazu die Polynomdivision und dividieren das Polynom $f(x)$ durch $g(x)$:

$$\begin{array}{rl}
2x^5 + x^4 + 4x + 3 & : (3x^2 + 1) = 4x^3 + 2x^2 + 2x + 1 \\
\underline{-2x^5 + x^3} & \\
x^4 + x^3 & \\
\underline{-x^4 + 3x^2} & \\
x^3 + 3x^2 & \\
\underline{-x^3 + 3x} & \\
3x^2 + 2x & \\
\underline{-3x^2 + 4} & \\
2x + 2 &
\end{array}$$

Check:

$$(4x^3 + 2x^2 + 2x + 1) \cdot (3x^2 + 1) + (2x + 2) = 2x^5 + x^4 + 4x + 3.$$

Damit haben wir das Polynom $f(x)$ geschrieben in der Form:

$$f(x) = q(x) \cdot g(x) + r(x)$$

mit:

$$\begin{aligned} q(x) &= 4x^3 + 2x^2 + 2x + 1 \\ r(x) &= 2x + 2. \end{aligned}$$

Ein Polynom $g \in \mathtt{GF}(p)[x]$ teilt das Polynom $f \in \mathtt{GF}(p)[x]$ falls ein Polynom $h \in \mathtt{GF}(p)[x]$ existiert, so dass $f = gh$. Man sagt auch, dass g der **Teiler** von f ist, oder dass f ein **Vielfaches** von g ist oder dass f *teilbar* durch g ist.

22.2.3 Euklidischer Algorithmus für Polynome

Wir betrachten in diesem Abschnitt Polynome über dem Körper der reellen Zahlen $\mathbb{R}$. Die folgenden Darstellungen können jedoch auch auf Polynome mit Koeffizienten über irgendeinem beliebigen Körper angewendet werden. Sind f und g zwei Polynome aus $\mathbb{R}[x]$, dann sagt man, $f \mid g$, falls es ein Polynom $h \in \mathbb{R}[x]$ gibt, so dass $g = f \cdot h$.

Man definiert den größten gemeinsamen Teiler zweier Polynome $\texttt{ggT}(f, g)$ in der gleichen Art und Weise wie für ganze Zahlen. Der ggT zweier Polynome $f, g \in \mathbb{R}[x]$ ist das Polynom mit dem höchsten Grad, welches sowohl f als auch g teilt.

Das in dieser Weise definierte Polynom ist nicht eindeutig, da man ein anderes Polynom mit gleichem Grad einfach dadurch erhält, indem man das Polynom mit einem konstanten Faktor multipliziert. Man kann den ggT aber eindeutig machen, indem man fordert, dass das ggT–Polynom normalisiert ist.[6]

Man sagt, die beiden Polynome $f, g \in \mathbb{R}[x]$ sind *relative prime* Polynome, wenn das ggT Polynom aus dem konstanten Polynom 1 besteht.

Wenn der größte gemeinsame Teiler zweier Polynome $f, g \in \mathbb{R}[x]$ berechnet wird, berechnet man de facto zwei Polynome $u, v \in \mathbb{R}[x]$ mit:

$$\texttt{ggT}(f, g)(x) = u(x) \cdot f(x) + v(x) \cdot g(x).$$

Wie für ganze Zahlen kann der größte gemeinsame Teiler zweier Polynome $f, g \in \mathcal{F}[x]$ mit Hilfe des **Euklidischen Algorithmus** berechnet werden. Ohne Einschränkung der Allgemeinheit nehmen wir an, dass $g \neq 0$ und g teilt nicht das Polynom f. Dann benutzen wir den Divisionsalgorithmus iterativ in der folgenden Weise:

$$\begin{aligned}
f &= q_1 g + r_1 && 0 \leq \deg(r_1) < \deg(g) \\
g &= q_2 r_1 + r_2 && 0 \leq \deg(r_2) < \deg(r_1) \\
r_1 &= q_3 r_4 + r_3 && 0 \leq \deg(r_3) < \deg(r_2) \\
&\vdots && \vdots \\
r_{s-2} &= q_s r_{s-1} + r_s && 0 \leq \deg(r_s) < \deg(r_{s-1}) \\
& && r_{s-1} = q_{s+1} r_s.
\end{aligned}$$

Hierbei sind $q_1, \ldots, q_{s+1}$ und $r_1, \ldots, r_s$ Polynome in $F[x]$. Da $\deg(g)$ endlich ist, muß das Verfahren nach endlich vielen Schritten terminieren. Falls der letzte, nichtverschwindende Rest r_s den führenden Koeffizienten b hat, dann ist der größte gemeinsame Teiler der beiden Polynome f, g durch $\texttt{ggT}(f, g) = b^{-1} \cdot r_s$ gegeben.

Beispiel [22.59]

In einem ersten Beispiel ([129]) berechnen wir den ggT der beiden Polynome

[6] Dies impliziert, dass der Koeffizient der höchsten Potenz den Wert 1 hat.

über $\mathbb{R}$

$$f(x) = x^4 + x^2 + 1$$
$$g(x) = x^2 + 1$$

In einem ersten Schritt dividieren wir f durch g und erhalten:

$$\begin{array}{l} x^4 + x^2 + 1 : x^2 + 1 = x^2 \\ \underline{-x^4 - x^2} \\ \qquad\quad +1 \end{array}$$

Daher

$$f(x) = q_1(x) \cdot g(x) + r_1(x) = x^2 \cdot (x^2 + 1) + 1$$

und wir erhalten:

$$q_1(x) = x^2$$
$$r_1(x) = 1$$

Gemäß dem EUKLIDischen Algorithmus müssen wir im nächsten Schritt berechnen

$$g = q_2 \cdot r_1 + r_2$$

Diviert man daher $g = x^2 + 1$ durch $r_1 = 1$ erhält man g; daher

$$q_2(x) = x^2 + 1$$
$$r_2(x) = 0$$

und der größte gemeinsame Teiler ist $r_1 = 1$. Daher sind die beiden Polynome coprim und wir können setzen:

$$\mathtt{ggT}(f, g) = 1 = 1 \cdot (x^4 + x^2 + 1) - x^2 \cdot (x^2 - 1)$$

Beispiel [22.60] Um den EUKLIDischen Algorithmus für einen etwas komplexeren Fall zu demonstrieren, berechnen wir den ggT der folgenden beiden Polynome über $\mathbb{R}[x]$:

$$f(x) = x^4 - 4x^3 + 6x^2 - 4x + 1$$
$$g(x) = x^3 - x^2 + x + 1$$

In einem ersten Schritt dividieren wir das Polynom f durch das Polynom g:

$$\begin{array}{llllll} +x^4 & -4x^3 & +6x^2 & -4x & +1 & : (x^3 - x^2 + x + 1) = x - 3 \\ -x^4 & +x^3 & -x^2 & +x & & \\ \hline & -3x^3 & +5x^2 & -3x & +1 & \\ & 3x^3 & -3x^2 & +3x & -3 & \\ \hline & & 2x^2 & & -2 & \end{array}$$

Also:

$$f(x) = q_1(x) \cdot g(x) + r_1(x) = (x-3) \cdot (x^3 - x^2 + x + 1) + (2x^2 - 2)$$

i.e., wir erhalten:

$$q_1(x) = x - 3$$
$$r_1(x) = 2x^2 - 2$$

Der zweite Schritt besteht darin, eine Zerlegung von g in der Form

$$g(x) = q_2 r_1 + r_2$$

zu finden. Das heißt wur müssen das Polynom $g(x)$ durch den Rest der Division im ersten Schritt teilen.

$$\begin{array}{llll} x^3 & -x^2 & +x & -1 \quad : (2x^2 - 2) = \frac{1}{2}x - \frac{1}{2} \\ -x^3 & & +x & \\ \hline & -x^2 & +2x & -1 \\ & +x^2 & & -1 \\ \hline & & 2x & -2 \end{array}$$

mit erhalten wir:

$$g(x) = q_2(x) \cdot r_1(x) + r_2(x) = (1/2x - 1/2) \cdot (2x^2 - 2) + (2x - 2)$$

und:

$$q_2(x) = 1/2x - 1/2$$
$$r_2(x) = 2x^2 - 2$$

Im dritten Schritt suchen wir die Zerlegung:

$$r_1(x) = q_3 r_2 + r_3.$$

Teilen wir das Polynom r_1 durch r_2, dann erhalten wir:

$$(2x^2 - 2) : (2x - 2) = x + 1$$

ohne Rest. Daher

$$r_1(x) = (x+1)r_2 = (x+1)(2x-2).$$

Damit erhalten wir das Resultat: $\mathrm{ggT}(f,g) = x - 1$, i.e. die normierte Form von $2x - 2$.

Um den größten gemeinsamen Teiler von f und g in der Form

$$\mathrm{ggT}(f,g) = u \cdot f + v \cdot g$$

zu schreiben, betrachten wir:

$$\begin{aligned} \mathrm{ggT}(f,g) = \frac{1}{2}r_2 &= \frac{1}{2}(g - q_2 \cdot r_1) \\ &= \frac{1}{2}\Big(g - q_2 \cdot (f - q_1 \cdot g)\Big) \\ &= \frac{1}{2}\Big(g - q_2 \cdot f + q_1 \cdot q_2 \cdot g\Big) \\ &= -\frac{1}{2}q_2 f + \frac{1}{2}(1 + q_1 q_2) \cdot g. \end{aligned}$$

Nun ist

$$\begin{aligned} q_1 \cdot q_2 &= \frac{1}{2}(x-3)(x-1) \\ &= \frac{x^2}{2} - 2 + \frac{3}{2}. \end{aligned}$$

Damit erhalten wir:

$$\begin{aligned} \mathrm{ggT}(f,g) &= -\frac{1}{2}q_2 f + \frac{1}{2}(1 + \frac{x^2}{2} - 2 + \frac{3}{2}) \cdot g \\ &= \big(-\frac{x}{4} + \frac{1}{4}\big) f + \big(\frac{x^2}{4} - x + \frac{5}{4}\big) g \end{aligned}$$

22.2.4 Erweiterter Euklidischer Algorithmus

Wie für ganze Zahlen gibt es auch für Polynome eine erweiterte Version des EUKLIDischen Algorithmus. Dieser wird hauptsächlich dafür verwendet, inverse Polynome zu berechnen.

Erweiterter Euklidischer Algorithmus für Polynome

S0 Input
Zwei Polynome $f(x)$ und $g(x)$

S1 Fall $g = 0$
Falls $g(x) = 0$ setze

$$d(x) \leftarrow h(x) \quad s(x) \leftarrow 1 \quad t(x) \leftarrow 0$$

und: Return $d(x), s(x), t(x) \quad \longrightarrow \quad$ STOP

S2 Initialisierung
Setze

$$s_2(x) \leftarrow 1 \quad s_1(x) \leftarrow 0 \quad t_2(x) \leftarrow 0 \quad t_1(x) \leftarrow 1$$

S3 Berechne
Solange $g(x) \neq 0$ führe folgende Schritte aus:

S3-a Dividiere f durch g und finde den Rest $r(x)$

$$q(x) \leftarrow f(x) \text{ div } g(x) \qquad r(x) \leftarrow f(x) - g(x) \cdot q(x)$$

S3-b Berechne

$$s(x) \leftarrow s_2(x) - q(x)s_1(x) \qquad t(x) \leftarrow t_2(x) - q(x) \cdot t_1(x)$$

S3-c Ersetze $f(x) \leftarrow g(x) \quad g(x) \leftarrow r(x)$

S3-d Ersetze

$$s_2(x) \leftarrow s_1(x) \quad s_1(x) \leftarrow s(x)$$
$$t_2(x) \leftarrow t_1(x) \quad t_1(x) \leftarrow t(x)$$

S4 Ersetzung
Setze $d(x) \leftarrow f(x) \quad s(x) \leftarrow s_2(x) \quad t(x) \leftarrow t_2(x)$

S5 Output
Return $d(x), s(x), t(x) \quad \longrightarrow \quad$ STOP

Der Input des obigen Algorithmus sind zwei Polynome f und g. Die Ausgabe dieses Algorithmus sind drei Polynome:

- $d(x) \quad \Longrightarrow \quad$ dies ist der größte gemeinsame Teiler von $f(x)$ und $g(x)$

– zwei Polynome $s(x), t(x)$, die die Bedingung

$$s(x)f(x) + t(x)g(x) = d(x)$$

erfüllen.

Sind daher die beiden Poynome f und g coprim, dann gilt:

$$s(x)f(x) + t(x)g(x) = 1$$

Der erweiterte EUKLIDische Algorithmus wird verwendet, inverse Polynome zu berechnen. Dies betrachten wir anhand von Beispielen.

Beispiel [22.61]

Als ein erstes Beispiel betrachten wir nochmals die beiden Polynome in $\mathbb{R}[x]$:

$$f(x) = x^4 - 4x^3 + 6x^2 - 4x + 1,$$
$$g(x) = x^3 - x^2 + x + 1.$$

Step1: S0 $\longrightarrow$ Input: Die beiden Polynome f, g

Step2: S1 $\longrightarrow$ Check: $g = 0$? Da $g \neq 0$ weiter mit Schritt S2.

Step3: S2 $\longrightarrow$ Initialisierung:
Setze

$$s_2(x) \leftarrow 1 \quad s_1(x) \leftarrow 0 \quad t_2(x) \leftarrow 0 \quad t_1(x) \leftarrow 1$$

Step4: S3: $\longrightarrow$ Solange $g \neq 0$ führe folgende Schritte durch:

Step5: S3a: Dividiere f durch g und berechne den Rest:

$$q(x) = f(x) : g(x) = x - 3$$
$$r(x) = f(x) + q(x) \cdot g(x) = 2x^2 - 2.$$

Daher setzen wir:

$$q(x) \longleftarrow x - 3$$
$$r(x) \longleftarrow 2x^2 - 2.$$

Step6: S3b: In einem nächsten Schritt berechnen und setzen wir:

$$s \leftarrow s_2 - qs_1 = 1$$
$$t \leftarrow t_2 - qt_1 = -(x - 3).$$

Step7: S3c: Wir substituieren:

$$f \leftarrow g = x^3 - x^2 + x + 1$$
$$g \leftarrow r = 2x^2 - 2.$$

Step8: S3d: Variablenersetzung:

$$\begin{aligned} s_2 &\leftarrow s_1 = 0 \\ s_1 &\leftarrow s = 1 \\ t_2 &\leftarrow t_1 = 1 \\ t_1 &\leftarrow t = -(x-3). \end{aligned}$$

Am Ende der ersten Iteration haben wir also folgende Werte für die Variablen:

$$\begin{aligned} f &= x^3 - x^2 + x + 1 \\ g &= 2x^2 - 2 \\ s &= 1 \\ t &= -(x-3) \\ s_2 &= 0 \\ s_1 &= 1 \\ t_2 &= 1 \\ t_1 &= -(x-3). \end{aligned}$$

Step9: Da $g \neq 0$ führen wir Schritt S3a erneut aus, teiel also f durch g und berechnen den Rest:

$$\begin{aligned} q(x) &= f(x) : g(x) = \frac{1}{2}(x-1) \\ r(x) &= f(x) + q(x) \cdot g(x) = 2x - 2. \end{aligned}$$

Wir setzen also:

$$\begin{aligned} q(x) &\longleftarrow \frac{1}{2}(x-1) \\ r(x) &\longleftarrow 2x - 2. \end{aligned}$$

Step10: S3b: Im nächsten Schritt berechnen wir:

$$\begin{aligned} s &\leftarrow s_2 - q s_1 = -\frac{1}{2}(x-1) \\ t &\leftarrow t_2 - q t_1 = 1 - \frac{1}{2}(x-1)(-1)(x-3) = \frac{x^2}{2} - 2x + \frac{5}{2}. \end{aligned}$$

Step11: S3c: Wir ersetzen:

$$\begin{aligned} f &\leftarrow g = 2x^2 - 2 \\ g &\leftarrow r = 2x - 2 \end{aligned}$$

Step12: S3d: Variablenersetzung führt auf:

$$\begin{aligned} s_2 &\leftarrow s_1 = 1 \\ s_1 &\leftarrow s = -\frac{1}{2}(x-1) \\ t_2 &\leftarrow t_1 = -(x-3) \\ t_1 &\leftarrow t = \frac{x^2}{2} - 2x + \frac{5}{2}. \end{aligned}$$

Am Ende des zweiten Iterationsschritts haben wir also die Werte:

$$\begin{aligned} f &= 2x^2 - 2 & g &= 2x - 2 \\ s &= -\frac{1}{2}(x-1) & t &= \frac{x^2}{2} - 2x + \frac{5}{2} \\ s_2 &= 1 & s_1 &= -\frac{1}{2}(x-1) \\ t_2 &= -(x-3) & t_1 &= \frac{x^2}{2} - 2x + \frac{5}{2}. \end{aligned}$$

Step13: Da $g \neq 0$ führen wir Schritt S3a erneut aus indem f durch g geteilt und der Rest bestimmt wird:

$$\begin{aligned} q(x) &= f(x) : g(x) = x + 1 \\ r(x) &= f(x) + q(x) \cdot g(x) = 0. \end{aligned}$$

Daher setzen wir:

$$\begin{aligned} q(x) &\longleftarrow x + 1 \\ r(x) &\longleftarrow 0. \end{aligned}$$

Step14: S3b: Wir berechnen und setzen:

$$s \leftarrow s_2 - qs_1 = 1 - (x+1)\frac{-1}{2}(x-1) = \frac{1}{2}(x^2+1)$$

$$t \leftarrow t_2 - qt_1 = -(x-3) - (x+1)(x^2/2 - 2x + 5/2) = -\frac{x^3}{2} + \frac{3}{2}x^2 - \frac{3}{2}x + \frac{1}{2}.$$

Step15: S3c: Ersetzung:

$$\begin{aligned} f &\leftarrow g = 2x - 2 \\ g &\leftarrow r = 0. \end{aligned}$$

Step16: S3d: Variablensubstitution ergibt:

$$\begin{aligned} s_2 &\leftarrow s_1 = -\frac{1}{2}(x-1) \\ s_1 &\leftarrow s = \frac{1}{2}(x^2+1) \\ t_2 &\leftarrow t_1 = \frac{x^2}{2} - 2x + \frac{5}{2} \\ t_1 &\leftarrow t = -\frac{x^3}{2} + \frac{3}{2}x^2 - \frac{3}{2}x + \frac{1}{2}. \end{aligned}$$

Damit haben wir am Ende der dritten Iteration die folgenden Werte:

$$\begin{aligned} f &= 2x - 2 & g &= 0 \\ s &= \frac{1}{2}(x^2+1) & t &= -\frac{x^3}{2} + \frac{3}{2}x^2 - \frac{3}{2}x + \frac{1}{2} \\ s_2 &= -\frac{1}{2}(x-1) & s_1 &= \frac{1}{2}(x^2+1) \\ t_2 &= \frac{x^2}{2} - 2x + \frac{5}{2} & t_1 &= -\frac{x^3}{2} + \frac{3}{2}x^2 - \frac{3}{2}x + \frac{1}{2}. \end{aligned}$$

Step17: S3: Da $g = 0$ endet die Schleife und wir fahren mit Schritt **S4** fort:
Wir setzen:

$$\begin{aligned} d(x) &\longleftarrow f(x) = 2(x-1) \\ s(x) &\longleftarrow s_2(x) = -\frac{1}{2}(x-1) \\ t(x) &\longleftarrow t_2(x) = \frac{x^2}{2} - 2x + \frac{5}{2}. \end{aligned}$$

Step18: **S5**: Output: Der Algorithmus gibt folgende Werte zurück:

$$\begin{aligned} d(x) &= 2(x-1) \\ s(x) &= -\frac{1}{2}(x-1) \\ t(x) &= \frac{x^2}{2} - 2x + \frac{5}{2}. \end{aligned}$$

Wir haben somit das frühere Resultat erneut abgeleitet:

$$d(x) = s(x) \cdot f(x) + t(x) \cdot g(x)$$

mit:

$$x - 1 = -\frac{1}{4}(x-1) \cdot f(x) + (\frac{x^2}{4} - x + \frac{5}{4}) \cdot g(x).$$

Beispiel [22.62]

In einem zweiten Beispiel — dieses Beispiel ist aus [154] — berechnen wir den ggT der beiden Polynome

$$\begin{aligned} f(x) &= x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + 1 \\ g(x) &= x^9 + x^6 + x^5 + x^3 + x^2 + x + 1 \end{aligned}$$

Der zugrundeliegende Körper ist $\mathtt{GF}(2)$.

Step1: S0 $\longrightarrow$ Input: Die beiden Polynome f und g.

Step2: S1 $\longrightarrow$ Check: $g = 0$? Da $g \neq 0$ weiter mit Schritt S2.

Step3: S2 $\longrightarrow$ Initialisierung:
Wir setzen:

$$s_2(x) \leftarrow 1 \quad s_1(x) \leftarrow 0 \quad t_2(x) \leftarrow 0 \quad t_1(x) \leftarrow 1.$$

Step4: S3: $\longrightarrow$ Solange $g \neq 0$ führe die folgenden Schritte aus:

Step5: S3a: Dividiere f durch g berechne den Rest:

$$q(x) = f(x) : g(x) = x + 1$$
$$r(x) = f(x) + q(x) \cdot g(x) = x^8 + x^7 + x^6 + x^2 + x.$$

Daher setzen wir:

$$q(x) \longleftarrow x + 1$$
$$r(x) \longleftarrow x^8 + x^7 + x^6 + x^2 + x.$$

Step6: S3b: Berechne

$$s \leftarrow s_2 - qs_1 = 1$$
$$t \leftarrow t_2 - qt_1 = x + 1.$$

Wir arbeiten in dem Körper $\mathbb{Z}_2$.

Step7: S3c: Wir ersetzen:

$$f(x) \leftarrow g(x) = x^9 + x^6 + x^5 + x^3 + x^2 + x + 1$$
$$g(x) \leftarrow r(x) = x^8 + x^7 + x^6 + x^2 + x.$$

Step8: S3d: Variablensubstitution:

$$s_2 \leftarrow s_1 = 0 \quad s_1 \leftarrow s = 1$$
$$t_2 \leftarrow t_1 = 1 \quad t_1 \leftarrow t = x + 1.$$

Am Ende des ersten Durchlaufs haben wir also folgende Werte:

$$f = x^9 + x^6 + x^5 + x^3 + x^2 + x + 1$$
$$g = x^8 + x^7 + x^6 + x^2 + x$$
$$s = 1 \qquad t = x + 1$$
$$s_2 = 0 \qquad s_1 = 1$$
$$t_2 = 1 \qquad t_1 = x + 1$$

Step9: Da $g \neq 0$ führen wir den Schritt S3a erneut aus, dividieren f durch g und berechen den Rest:

$$q(x) = f(x) : g(x) = x + 1$$
$$r(x) = f(x) + q(x) \cdot g(x) = x^5 + x^2 + x + 1.$$

Wir setzen

$$q(x) \longleftarrow x + 1$$
$$r(x) \longleftarrow x^5 + x^2 + x + 1.$$

Step10: S3b: Berechne

$$s \leftarrow s_2 - qs_1 = x + 1$$
$$t \leftarrow t_2 - qt_1 = 1 - (x + 1)(x + 1) = x^2.$$

Step11: S3c: Ersetzen:

$$f \leftarrow g = x^8 + x^7 + x^6 + x^2 + x$$
$$g \leftarrow r = x^5 + x^2 + x + 1.$$

Step12: S3d: Variablensubstitution:

$$s_2 \leftarrow s_1 = 1$$
$$s_1 \leftarrow s = x + 1$$
$$t_2 \leftarrow t_1 = x + 1$$
$$t_1 \leftarrow t = x^2.$$

Die zweite iteraation liefert also folgende Werte:

$$f = x^8 + x^7 + x^6 + x^2 + x \qquad g = x^5 + x^2 + x + 1$$
$$s = x + 1 \qquad t = x^2$$
$$s_2 = 1 \qquad s_1 = x + 1$$
$$t_2 = x + 1 \qquad t_1 = x^2.$$

Step13: Da $g \neq 0$ muß Schritt S3a erneut ausgeführt werden. Dividiere f durch g berechne den Rest:

$$q(x) = f(x) : g(x) = x^3 + x^2 + x + 1$$
$$r(x) = f(x) + q(x) \cdot g(x) = x^3 + x + 1.$$

Setze

$$q(x) \longleftarrow x^3 + x^2 + x + 1$$
$$r(x) \longleftarrow x^3 + x + 1.$$

Step14: S3b: Berechne

$$s \leftarrow s_2 - qs_1 = 1 + (x^3 + x^2 + x + 1)(x + 1) = x^4$$
$$t \leftarrow t_2 - qt_1 = x + 1 + (x^3 + x^2 + x + 1)x^2 = x^5 + x^4 + x^3 + x^2 + x + 1.$$

Step15: S3c: Ersetze:

$$f \leftarrow g = x^5 + x^2 + x + 1$$
$$g \leftarrow r = x^3 + x + 1$$

Step16: S3d: Variablensubstitution:

$$s_2 \leftarrow s_1 = x + 1$$
$$s_1 \leftarrow s = x^4$$
$$t_2 \leftarrow t_1 = x^2$$
$$t_1 \leftarrow t = x^5 + x^4 + x^3 + x^2 + x + 1$$

Der dritte Durchlauf liefert also:

$$\begin{aligned} f = x^5 + x^2 + x + 1 \qquad & g = x^3 + x + 1 \\ s = x^4 \qquad & t = x^5 + x^4 + x^3 + x^2 + x + 1 \\ s_2 = x + 1 \qquad & s_1 = x^4 \\ t_2 = x^2 \qquad & t_1 = x^5 + x^4 + x^3 + x^2 + x + 1. \end{aligned}$$

Step17: Da $g \neq 0$ wird Schritt S3a erneut ausgeführt:

$$\begin{aligned} q(x) &= f(x) : g(x) = x^2 + x + 1 \\ r(x) &= f(x) + q(x) \cdot g(x) = 0. \end{aligned}$$

Setze

$$\begin{aligned} q(x) &\longleftarrow x^2 + x + 1 \\ r(x) &\longleftarrow 0 \end{aligned}$$

Step18: S3b: Berechne und setze:

$$\begin{aligned} s \leftarrow s_2 - qs_1 &= x + 1 + (x^2 + 1)x^4 = x^6 + x^4 + x + 1 \\ t \leftarrow t_2 - qt_1 &= x^2 + (x^2 + 1)(x^5 + x^4 + x^3 + x^2 + x + 1)x^2 \\ &= x^7 + x^6 + x^2 + x + 1. \end{aligned}$$

Step19: S3c: Ersetzen führt auf:

$$\begin{aligned} f &\leftarrow g = x^3 + x + 1 \\ g &\leftarrow r = 0 \end{aligned}$$

Step20: S3d: Variablensubstitution:

$$\begin{aligned} s_2 &\leftarrow s_1 = x^4 \\ s_1 &\leftarrow s = x^6 + x^4 + x + 1 \\ t_2 &\leftarrow t_1 = x^5 + x^4 + x^3 + x^2 + x + 1 \\ t_1 &\leftarrow t = x^7 + x^6 + x^2 + x + 1. \end{aligned}$$

Damit ist am Ende des vierten Durchlaufs:

$$\begin{aligned} f = x^3 + x + 1 \qquad & g = 0 \\ s = x^6 + x^4 + x + 1 \qquad & t = x^7 + x^6 + x^2 + x + 1 \\ s_2 = x^4 \qquad & s_1 = x^6 + x^4 + x + 1 \\ t_2 = x^5 + x^4 + x^3 + x^2 + x + 1 \qquad & t_1 = x^7 + x^8 + x^2 + x + 1 \end{aligned}$$

Da g verschwindet, terminiert die Schleife und wir erhalten:

$$\begin{aligned} d(x) &\leftarrow f(x) = x^3 + x + 1 \\ s(x) &\leftarrow s_2(x) = x^4 \\ t(x) &\leftarrow t_2(x) = x^5 + x^4 + x^3 + x^2 + x + 1. \end{aligned}$$

Daher ist:

$$\texttt{ggT}(f(x), g(x)) = x^3 + x + 1$$

und man kann direkt nachrechnen, dass gilt:

$$x^3 + x + 1 = x^4 \cdot f(x) + (x^5 + x^4 + x^3 + x^2 + x + 1) \cdot g(x).$$

22.2.5 Irreduzible Polynome

Ein Polynom $f(x)$ über einem Körper $GF(p)$ heisst **irreduzibel** genau dann, wenn $f(x)$ nicht ausgedrückt werden kann als Produkt zweier anderer Polynome über diesem Körper, die beide kleineren Grad haben als $f(x)$. In Analogie zu ganzen Zahlen nennt man irreduzible Polynome auch **Prim–Polynome**.

Beispiel [22.63]

Das Polynom

$$f(x) = x^4 + 1$$

über $GF(2)$ ist reduzibel, da

$$f(x) = x^4 + 1 = (x + 1) \cdot (x^3 + x^2 + x + 1).$$

Beispiel [22.64]

Wir betrachten das Polynom

$$f(x) = x^3 + x + 1$$

über $\texttt{GF}(2)$. Man sieht direkt wegen dem konstanten Faktor 1, dass x kein Faktor von $f(x)$ sein kann. Man zeigt durch Polynomdivision, dass $x+1$ ebenfalls kein Faktor von $f(x)$ ist:

$$
\begin{array}{l}
(x^3 + x + 1) : (x + 1) = x^2 + x \\
x^3 + x^2 \\
\hline
\quad x^2 + x + 1 \\
\quad x^2 + x \\
\hline
\qquad\quad 1
\end{array}
$$

Daher hat $f(x)$ keinen Faktor vom Grad 1. Das Polynom $f(x)$ ist ein Polynom vom Grad 3. Wenn $f(x)$ reduzibel ist, dann muss dieses Polynom einen Faktor haben vom Grad 2 und einen Faktor vom Grad 1. Da wir gezeigt haben, dass kein Polynom vom Grad 1 — also x und $x + 1$ — das Polynom $f(x)$ ohne Rest teilt, ist $f(x)$ irreduzibel über dem Körper $\mathtt{GF}(2)$.

Die Primelemente des Rings $F[x]$ — wenn man den Ring der ganzen Zahlen betrachtet sind dies gerade die Primzahlen — nennt man üblicherweise **irreduziblen Polynome**.

Definition [22.23]:

Ein Polynom $p \in F[x]$ heißt **irreduzibel über dem Körper** F oder *irreduzibel in* $F[x]$ oder *prim in* $F[x]$, falls p positiven Grad hat und $p = b \cdot c,\ b, c \in F[x]$ impliziert, dass entweder b oder c ein konstantes Polynom ist.

Kurz gesagt, ein Polynom ist irreduzibel über einem Körper F, falls dieses Polynom nur die triviale Faktorisierung erlaubt. Ein Polynom in $F[x]$ mit positivem Grad, das nicht irreduzibel über dem Körper F ist, nennt man **reduzibel**.

Die Reduzibilität oder Irreduzibilität eines Polynomes hängt entscheidend davon ab, über welchem Körper man das Polynom betrachtet. Beispielsweise ist das Polynom

$$f(x) = x^2 - 2$$

irreduzibel über dem Körper der rationalen Zahlen $\mathbb{Q}$, jedoch reduzibel über den reellen Zahlen $\mathbb{R}$, da über den reellen Zahlen dieses Polynom geschrieben werden kann in der Form:

$$f(x) = (x + \sqrt{2}) \cdot (x - \sqrt{2}).$$

Irreduzible Polynome sind von entscheidender Bedeutung für die Struktur des Polynomrings $GF[x]$, da alle Polynome in $GF[x]$ in (fast) eindeutiger Weise als Produkte von irreduziblen Polynomen geschrieben werden können. Daher gilt folgender Satz:[7]

Theorem [14]:

Jedes Polynom $f \in F[x]$ mit positivem Grad kann in der Form

$$f = a p_1^{e_1} \cdots p_k^{e_k}$$

geschrieben werden. Dabei ist $a \in F$, die $p_1, p_2, \ldots, p_k$ sind unterschiedliche normierte irreduzible Polynome in $F[x]$ und $e_1, e_2, \ldots, e_k$ sind positive ganze Zahlen. Weiterhin ist diese Faktorisierung eindeutig bis auf die Reihenfolge, in der die Faktoren auftreten.

Beweis: $\longrightarrow$ Lidl, Niederreiter [144], pg. 24.

Diese Zerlegung eines beliebigen Polynoms in ihre irreduziblen Bestandteile nennt man **kanonische Faktorisierung** des Polynomes f in $F[x]$.

Eine zentrale Frage über Polynome in $F[x]$ ist, ob ein gegebenes Polynom reduzibel oder irreduzibel über einem Körper F ist. Für unsere Zwecke sind insbesondere irreduzible Polynome über dem Körper $\mathtt{GF}(p)$ interessant. Um sämtliche irreduziblen Polynome mit einem festen Grad n über $\mathtt{GF}(p)$ zu finden, kann man zunächst alle reduziblen normierten Polynome vom Grad n über $\mathtt{GF}(p)$ berechnen und anschließend eleminiert man diese aus der Menge aller normierten Polynomen über $\mathtt{GF}(p)$. Falls p oder n große Werte annehmen, ist diese Methode jedoch nicht effektiv umsetzbar.

Beispiel [22.65]

Wir suchen alle irreduziblen Polynome[8] über $\mathtt{GF}(2)$ vom Grad 4. Wir listen zunächst alle möglichen Polynome über $\mathtt{GF}(2)$ vom Grad 4 auf. Dafür gibt es

[7] Dieser Satz entspricht dem Fundamentalsatz der Zahlentheorie.
[8] Siehe [144], Example 1.60.

$2^4 = 16$ Möglichkeiten:

$$\begin{aligned}
p_1 &= x^4 \\
p_2 &= x^4 + x^3 \\
p_3 &= x^4 + x^2 \\
p_4 &= x^4 + x \\
p_5 &= x^4 + 1 \\
p_6 &= x^4 + x^3 + x^2 \\
p_7 &= x^4 + x^3 + x \\
p_8 &= x^4 + x^2 + x \\
p_9 &= x^4 + x^3 + 1 \\
p_{10} &= x^4 + x^2 + 1 \\
p_{11} &= x^4 + x + 1 \\
p_{12} &= x^4 + x^3 + x^2 + x \\
p_{13} &= x^4 + x^3 + x^2 + 1 \\
p_{14} &= x^4 + x^3 + x + 1 \\
p_{15} &= x^4 + x^2 + x + 1 \\
p_{16} &= x^4 + x^3 + x^2 + x + 1
\end{aligned}$$

Solch ein Polynom ist reduzibel über `GF(2)` genau dann, wenn das Polynom ein Teilerpolynom vom Grad 1 oder 2 hat. Daher berechnen wir sämtliche Produkte der Form:

$$(a_0 + a_1x + a_2x^2 + x^3) \cdot (b_0 + x)$$

und

$$(a_0 + a_1x + x^2) \cdot (b_0 + b_1x + x^2).$$

mit $a_i, b_j \in GF(2)$. Wir erhalten:

$$\begin{aligned}
(a_0 + a_1x + a_2x^2 + x^3) \cdot (b_0 + x) = x^4 + (a_2 + b_0)x^3 + (a_2b_0 + a_1)x^2 \\
(a_0 + a_1b_0)x + a_0b_0
\end{aligned}$$

und

$$\begin{aligned}
(a_0 + a_1x + x^2) \cdot (b_0 + b_1x + x^2) = x^4 + (a_1 + b_1)x^3 + (a_0 + a_1b_1 + b_0)x^2 \\
+ (a_0b_1 + a_1b_0)x + a_0b_0.
\end{aligned}$$

Wir setzen:

$$P_1(x) = x^4 + (a_2 + b_0)x^3 + (a_2b_0 + a_1)x^2 + (a_0 + a_1b_0)x + a_0b_0$$

und

$$P_2(x) = x^4 + (a_1 + b_1)x^3 + (a_0 + a_1b_1 + b_0)x^2 + (a_0b_1 + a_1b_0)x + a_0b_0.$$

Nun werden alle möglichen Belegungen der Koeffizienten a_0, a_1, a_2, b_0 für das Polynom $P_1(x)$ durchgespielt:

1. Setzen wir die Koeffizienten $a_0 = 0, a_1 = 0, a_2 = 0, b_0 = 0 \implies$
$$P_1(x) = x^4 = p_1(x).$$

2. $a_0 = 0, a_1 = 0, a_2 = 0, b_0 = 1 \implies$
$$P_1(x) = x^4 + x^3 = p_2(x).$$

3. $a_0 = 0, a_1 = 0, a_2 = 1, b_0 = 0 \implies$
$$P_1(x) = x^4 + x^3 = p_2(x).$$

4. $a_0 = 0, a_1 = 0, a_2 = 1, b_0 = 1 \implies$
$$P_1(x) = x^4 + x^2 = p_3(x).$$

5. $a_0 = 0, a_1 = 1, a_2 = 0, b_0 = 0 \implies$
$$P_1(x) = x^4 + x^2 = p_3(x).$$

6. $a_0 = 0, a_1 = 1, a_2 = 0, b_0 = 1 \implies$
$$P_1(x) = x^4 + x^3 + x^2 + x = p_{12}(x).$$

7. $a_0 = 0, a_1 = 1, a_2 = 1, b_0 = 0 \implies$
$$P_1(x) = x^4 + x^3 + x^2 = p_6(x).$$

8. $a_0 = 0, a_1 = 1, a_2 = 1, b_0 = 1 \implies$
$$P_1(x) = x^4 + x = p_4(x).$$

9. $a_0 = 1, a_1 = a_2 = b_0 = 0 \implies$
$$P_1(x) = x^4 + x = p_4(x).$$

10. $a_0 = 1, a_1 = 0, a_2 = 0, b_0 = 1 \implies$
$$P_1(x) = x^4 + x^3 + x + 1 = p_{14}(x).$$

11. $a_0 = 1, a_1 = 0, a_2 = 1, b_0 = 0 \implies$
$$P_1(x) = x^4 + x^3 + x = p_7(x).$$

12. $a_0 = 1, a_1 = 1, a_2 = 0, b_0 = 0 \implies$
$$P_1(x) = x_4 + x^2 + x = p_8(x).$$

13. $a_0 = 1, a_1 = 0, a_2 = 1, b_0 = 1 \implies$
$$P_1(x) = x_4 + x^2 + x + 1 = p_{15}(x).$$

14. $a_0 = 1, a_1 = 1, a_2 = 1, b_0 = 0 \implies$

$$P_1(x) = x_4 + x^3 + x^2 + x = p_{12}(x).$$

15. $a_0 = 1, a_1 = 1, a_2 = 0, b_0 = 1 \implies$

$$P_1(x) = x_4 + x^3 + x^2 + 1 = p_{13}(x).$$

16. $a_0 = 1, a_1 = 1, a_2 = 1, b_0 = 1 \implies$

$$P_1(x) = x_4 + 1 = p_5(x).$$

Die gleiche Rechnung mit den unterschiedlichen Belegungen der Koeffizienten a_0, a_1, b_0 und b_1 für das Polynom $P_2(x)$ ergibt folgende Liste:

1. Setzen wir die Koeffizienten $a_0 = a_1 = b_0 = b_1 = 0 \implies$

$$P_2(x) = x^4 = p_1(x).$$

2. $a_0 = a_1 = 0, b_0 = 0, b_1 = 1 \implies$

$$P_2(x) = x^4 + x^3 = p_2(x).$$

3. $a_0 = a_1 = 0, b_0 = 1, b_1 = 0 \implies$

$$P_2(x) = x^4 + x^2 = p_3(x).$$

4. $a_0 = a_1 = 0, b_0 = 1; b_1 = 1 \implies$

$$P_2(x) = x^4 + x^3 + x^2 = p_6(x).$$

5. $a_0 = 0, a_1 = 1, b_0 = 0, b_1 = 0 \implies$

$$P_2(x) = x^4 + x^3 = p_2(x).$$

6. $a_0 = 0, a_1 = 1, b_0 = 0, b_1 = 1 \implies$

$$P_2(x) = x^4 + x^2 = p_3(x).$$

7. $a_0 = 0, a_1 = 1, b_0 = 1, b_1 = 0 \implies$

$$P_2(x) = x^4 + x^3 + x^2 + x = p_{12}(x).$$

8. $a_0 = 0, a_1 = 1, b_0 = 1, b_1 = 1 \implies$

$$P_2(x) = x^4 + x = p_4(x).$$

9. $a_0 = 1, a_1 = 0, b_0 = 0, b_1 = 0 \implies$

$$P_2(x) = x^4 + x^2 = p_3(x).$$

10. $a_0 = 1, a_1 = 0, b_0 = 0; b_1 = 1 \implies$
$$P_2(x) = x^4 + x^3 + x^2 + x = p_{12}(x).$$

11. $a_0 = 1, a_{=}0, b_0 = 1, b_1 = 0 \implies$
$$P_2(x) = x^4 + 1 = p_5(x).$$

12. $a_0 = 1, a_1 = 1, b_0 = 0, b_1 = 0 \implies$
$$P_2(x) = x_4 + x^3 + x^2 = p_6(x).$$

13. $a_0 = 1, a_1 = 0, b_0 = 1, b_1 = 1 \implies$
$$P_2(x) = x_4 + x^3 + x + 1 = p_{14}(x).$$

14. $a_0 = 1, a_1 = 1, b_0 = 1, b_1 = 0 \implies$
$$P_2(x) = x_4 + x^3 + x + 1 = p_{14}(x).$$

15. $a_0 = 1, a_1 = 1, b_0 = 0, b_1 = 1 \implies$
$$P_2(x) = x_4 + x = p_4(x).$$

16. $a_0 = 1, a_1 = 1, b_0 = 1, b_1 = 1 \implies$
$$P_2(x) = x_4 + x^2 + 1 = p_{10}(x).$$

Aus dieser Auswertung resultiert, dass die 13 Polynome

$$p_1(x), p_2(x), p_3(x), p_4(x), p_5(x), p_6(x), p_7(x),$$
$$p_8(x), p_{10}(x), p_{12}(x), p_{13}(x), p_{14}(x) \text{ und } p_{15}(x)$$

als Produkte von Polynomen 1. oder 2. Grades geschrieben werden können. Für die drei Polynome

$$p_9(x) = x^4 + x^3 + 1 \tag{22.4}$$
$$p_{11}(x) = x^4 + x + 1 \tag{22.5}$$
$$p_{16}(x) = x^4 + x^3 + x^2 + x + 1 \tag{22.6}$$

findet man keine Zerlegung in ein Polynom 1. oder 2. Grades. Daher sind $p_9(x), p_{11}(x)$ und $p_{16}(x)$ die drei irreduziblen Polynome 4. Grades über $\mathtt{GF}(2^4)$.

Im Kapitel [22.4] untersuchen wir die algebraische Struktur von $\mathtt{GF}(2^4)$ im Detail.

Theorem [15]:

Für $f \in \mathtt{GF}[x]$ ist der Restklassenring $\mathtt{GF}[x]/(f)$ ein Körper, genau dann und nur dann, wenn f irreduzibel über $\mathtt{GF}(p)$ ist.

Als Vorbereitung für das folgende Kapitel sehen wir uns die Struktur des Restklassenrings $\mathtt{GF}[x]/(f)$ näher an, wobei f ein beliebiges, nichtverschwindendes Polynom in $\mathtt{GF}[x]$ ist.

Der Restklassenring $\mathtt{GF}[x]/(f)$ besteht also aus Restklassen der Form

$$g + (f) = [g] \quad \text{mit } g \in \mathtt{GF}[x].$$

Die Operationen Addition und Multiplikation auf $\mathtt{GF}[x]/(f)$ sind wie üblich definiert. Zwei Restklassen $[g]$ und $[h]$ sind identisch, genau dann, wenn

$$g(x) \equiv h(x) \bmod f(x),$$

das heißt, wenn das Polynom $g - h$ durch das Polynom f (ohne Rest) teilbar ist. Dies ist gleichbedeutend mit der Forderung, dass g und h den gleichen Rest haben, wenn sie durch f geteilt werden. Jede Restklasse $g + (f)$ enthält einen eindeutigen Repräsentanten $r \in \mathtt{GF}[x]$ mit

$$\deg(r) < \deg(f),$$

was einfach der Rest der Division von g durch f ist. Den Prozess des Übergangs von g nach r nennt man **Reduktion modulo f.**.

Die unterschiedlichen Restklassen, die den Resklassenring $DF[x]/(f)$ aufbauen, können nun explizit angegeben werden:

> Der Restklassenring $GF[x]/(f)$ besteht aus den Klassen $r+(f)$ wobei r durch alle Polynome in $F[x]$ läuft mit $\deg(r) < \deg(f)$.

Ist $F = \mathbb{F}_p$ und der Grad des Polynoms f ist $n \geq 1$, dann ist die Anzahl der Elemente in $\mathbb{F}_p/(f)$ gleich der Anzahl der Polynome in $\mathbb{F}_p[x]$ vom Grad $< n$, was p^n ist.

Beispiel [22.66]

Sei

$$f(x) = x \in \mathtt{GF}[x].$$

Die $p^n = 2^1 = 2$ Polynome in $\mathtt{GF}(2)[x]$ mit Grad < 1 bestimmen alle Restklassen, die $\mathtt{GF}(2)[x]/(x)$ umfassen. Daher haben wir:

$$\mathtt{GF}(2)[x]/(x) = \{[0], [1]\} \cong \mathtt{GF}(2).$$

Beispiel [22.67]

Sei

$$f(x) = x^2 + x + 1 \in \mathtt{GF}(2)[x].$$

Der Restklassenring $\mathtt{GF}(2)[x]/(f)$ hat die $p^n = 2^2 = 4$ Elemente:

$$\mathtt{GF}(2)[x]/(x) = \{[0], [1], [x], [x+1]\}.$$

Auf diesem Restklassenring können wir die Addition (mod 2) ausführen, was auf folgende Verknüpfungstafel führt:

$+$	$[0]$	$[1]$	$[x]$	$[x+1]$
$[0]$	$[0]$	$[1]$	$[x]$	$[x+1]$
$[1]$	$[1]$	$[0]$	$[x+1]$	$[x]$
$[x]$	$[x]$	$[x+1]$	$[0]$	$[1]$
$[x+1]$	$[x+1]$	$[x]$	$[1]$	$[0]$

Die Multiplikation modulo f ergibt folgende Werte:

$\times$	$[0]$	$[1]$	$[x]$	$[x+1]$
$[0]$	$[0]$	$[0]$	$[0]$	$[0]$
$[1]$	$[0]$	$[1]$	$[x]$	$[x+1]$
$[x]$	$[0]$	$[x]$	$[x+1]$	$[1]$
$[x+1]$	$[0]$	$[x+1]$	$[1]$	$[x]$

Aus diese Tabellen — oder aus der Irreduzibilität von f über $\mathtt{GF}(2)$ und Theorem [15] folgt, dass $\mathtt{GF}(2)/(f)$ ein Körper ist. Man beachte, dies ist ein Beispiel eines Körpers, bei dem die Anzahl der Elemente keine Primzahl ist.

Beispiel [22.68]

Wir betrachten den Körper $GF(3)$ und das Polynom

$$f(x) = x^2 + 2 \in GF(3)[x].$$

Der Restklassenring $GF(3)/(f)$ wird aus $p^n = 3^2 = 9$ Polynomen aufgebaut:

$$GF(3)/(f) = \{[0], [1], [2], [x], [x+1], [x+2], [2x], [2x+1], [2x+2]\}.$$

Die Verknüpfungstabellen für $\mathbb{F}_3/(f)$ erzeugt man wieder durch die entsprechenden Polynom-Operationen mit anschließender Reduktion modulo f falls erforderlich.

$+$	$[0]$	$[1]$	$[2]$	$[x]$	$[x+1]$	$[x+2]$	$[2x]$	$[2x+1]$	$[2x+2$
$[0]$	$[0]$	$[1]$	$[2]$	$[x]$	$[x+1]$	$[x+2]$	$[2x]$	$[2x+1]$	$[2x+2$
$[1]$	$[1]$	$[2]$	$[0]$	$[x+1]$	$[x+2]$	$[x]$	$[2x+1]$	$[2x+2]$	$[2x]$
$[2]$	$[2]$	$[0]$	$[1]$	$[x+2]$	$[x]$	$[x+1]$	$[2x+2]$	$[2x]$	$[2x+1$
$[x]$	$[x]$	$[x+1]$	$[x+2]$	$[2x]$	$[2x+1]$	$[2x+2]$	$[0]$	$[1]$	$[2]$
$[x+1]$	$[x+1]$	$[x+2]$	$[x]$	$[2x+1]$	$[2x+2]$	$[2x]$	$[1]$	$[2]$	$[0]$
$[x+2]$	$[x+2]$	$[x]$	$[x+1]$	$[2x+2]$	$[2x]$	$[2x+1]$	$[2]$	$[0]$	$[1]$
$[2x]$	$[2x]$	$[2x+1]$	$[2x+2]$	$[0]$	$[1]$	$[2]$	$[x]$	$[x+1]$	$[x+2]$
$[2x+1]$	$[2x+1]$	$[2x+2]$	$[2x]$	$[1]$	$[2]$	$[0]$	$[x+1]$	$[x+2]$	$[x]$
$[2x+2]$	$[2x+2]$	$[2x]$	$[2x+1]$	$[2]$	$[0]$	$[1]$	$[x+2]$	$[x]$	$[x+1]$

Die Multiplikationstafel ergibt sich zu:

$\times$	$[0]$	$[1]$	$[2]$	$[x]$	$[x+1]$	$[x+2]$	$[2x]$	$[2x+1]$	$[2x+2]$
$[0]$	$[0]$	$[0]$	$[0]$	$[0]$	$[0]$	$[0]$	$[0]$	$[0]$	$[0]$
$[1]$	$[0]$	$[1]$	$[2]$	$[x]$	$[x+1]$	$[x+2]$	$[2x]$	$[2x+1]$	$[2x+2]$
$[2]$	$[0]$	$[2]$	$[1]$	$[2x]$	$[2x+2]$	$[2x+1]$	$[x]$	$[x+2]$	$[x+1]$
$[x]$	$[0]$	$[x]$	$[2x]$	$[1]$	$[x+1]$	$[2x+1]$	$[2]$	$[x+2]$	$[2x+2]$
$[x+1]$	$[0]$	$[x+1]$	$[2x+2]$	$[x+1]$	$[2x+2]$	$[0]$	$[2x+2]$	$[0]$	$[x+1]$
$[x+2]$	$[0]$	$[x+2]$	$[2x+1]$	$[2x+1]$	$[0]$	$[x+2]$	$[x+2]$	$[2x+1]$	$[0]$
$[2x]$	$[0]$	$[2x]$	$[x]$	$[2]$	$[2x+2]$	$[x+2]$	$[1]$	$[2x+1]$	$[x+1]$
$[2x+1]$	$[0]$	$[2x+1]$	$[x+2]$	$[x+2]$	$[0]$	$[2x+1]$	$[2x+1]$	$[x+2]$	$[0]$
$[2x+2]$	$[0]$	$[2x+2]$	$[x+1]$	$[2x+2]$	$[x+1]$	$[0]$	$[x+1]$	$[0]$	$[2x+2]$

Die Resultate erhält man beispielsweise wie folgt:

$$(2x+2) \times (x+1) = 2x^2 + 2x + 2x + 2 \equiv 2x^2 + x + 2$$

Reduktion modulo f führt auf:

$$\begin{array}{rl} 2x^2 + x + 2 & : (x^2+2) = 2 \\ \underline{-2x^2 - 1} & \\ x + 1 & \end{array}$$

Also:

$$(2x^2 + x + 2) : (x^2 + 2) = 2 \text{ Rest } x + 1$$

Daher ist

$$[2x+2] \times [x+1] = [x+1].$$

Aus diesen Tabellen erkennt man, dass $GF(3)/(f)$ keine Körperstruktur hat. Dies ist in Übereinstimmung mit dem Theorem [15], da das Polynom $f = x^2 + 2$ über $\mathbb{F}_3$ reduzibel ist:

$$x^2 + 2 = (x+1)(x+2).$$

Ist F ein beliebiger Körper und $f(x) \in F[x]$, dann führt die Ersetzung der Unbestimmten x in $f(x)$ durch ein festes Element aus F auf ein wohldefiniertes Element in F. Im DetaiL: Ist

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n \in F[x]$$

und $b \in F$, dann erhalten wir durch die Ersetzung von x durch b:

$$f(b) = a_0 + a_1 b + \cdots + a_n b^n \in F.$$

In jeder polynomialen Identität in $F[x]$ kann man ein festes $b \in F$ für x setzen und erhält eine gültige Identität in F.

Definition [22.24]:

Ein Element $b \in F$ heißt **Wurzel** (oder Nullstelle) des Polynoms $f \in F[x]$, wenn $f(b) = 0$.

Ein wichtiger Zusammenhang zwischen Wurzeln und der Teilbarkeit ist durch den folgenden Satz gegeben.

Theorem [16]:

Ein Element $b \in F$ ist eine Wurzel des Polynoms $f \in F[x]$ genau dann, wenn $x - b$ Teiler von $f(x)$ ist.

22.3 Endliche Körper der Form $\mathtt{GF}\left(2^n\right)$

Wir haben früher in diesem Kapitel gesagt, dass die Ordnung eines Körpers die Form p^n haben muss, wobei p eine Primzahl und n eine natürliche Zahl ist. Im Abschnitt [22.1] haben wir uns den speziellen Fall der endlichen Körper $\mathtt{GF}(p)$ angesehen. Wir haben gesehen, dass die modulare Arithmetik in $\mathbb{Z}_p$ alle Körperaxiome erfüllt. Für Polynome über p^n mit $n > 1$ führen die Operationen modulo p^n nicht auf einen Körper. in diesem Kapitel sehen wir, welche Strukturen die Axiome eines Körpers mit einer Menge von p^n Elementen erfüllen und sehen uns speziell den Körper $\mathtt{GF}\left(2^n\right)$ an.

22.3.1 Motivation

So ziemlich jeder Verschlüsselungsalgorithmus — symmetrisch oder asymmetrisch — verwendet arithmetische Operationen auf ganzen Zahlen. Ist eine dieser Operationen die Division, dann müssen wir notwendigerweise mit Arithmetik arbeiten, die über einem Körper definiert ist. Weiterhin arbeiten wir — aus Gründen der effizienten Implementierung solcher Algorithmen — mit ganzen Zahlen, die genau eine bestimmte Anzahl von Bits haben, typischerweise haben wir es mit ganzen Zahlen im Intervall 0 bis $2^n - 1$ zu tun, die durch n Bits dargestellt werden können.

Beispiel [22.69]

Angenommen man möchte einen konventionellen Verschlüsselungsalgorithmus entwerfen, der auf 8 Bit operiert und der Division ausführt. Mit 8 Bits können wir die Zahlen 0 bis 255 darstellen, das sind 256 Zahlen. Jedoch ist 256 keine Primzahl; führt man daher auf $\mathbb{Z}_{256}$ Arithmetik aus, ist diese Menge von ganzen Zahlen kein Körper, *i.e.* insbesondere gibt es Zahlen, die kein multiplikatives Inverses auf $\mathbb{Z}_{256}$ haben. Die Primzahl, die 256 am nächsten liegt, ist 251, dann hat $\mathbb{Z}_{251}$ mit der Arithmetik modulo 251 die algebraische Struktur eines Körpers. Verwendet man die 252 Zahlen in $\mathbb{Z}_{251}$, dann sind die 8–Bit Folgen, die die zahlen 252 bis 255 repräsentieren, ungenutzt, dies resultiert daher in einer ineffektiven Speichernutzung.

Das Beispiel [22.69] zeigt folgenden Sachverhalt. Sollen sämtliche arithmetischen Operationen genutzt werden und man will den kompletten Zahlenbereich nutzen, der durch n Bits dargestellt wird, dann funktioniert die Arithmetik modulo 2^n nicht. Äquivalent dazu ist die Aussage, die Menge der ganzen Zahlen $\mathbb{Z}_n$ mit der Arithmetik modulo 2^n bildet nicht die Struktur eines Körpers. Selbst wenn ein Verschlüsselungsalgorithmus nur Addition und Multiplikation verwendet (und keine Division) ist es fragwürdig, ob es sinnvoll ist, die Menge Z_{2^n} zu wählen.

Beispiel [22.70]

Angenommen, wir verwenden 3–Bit Blocks zur Verschlüsselung und verwenden

nur die Additions– und die Multiplikationsoperation. Dann ist die Arithmetik modulo 8 wohldefiniert. Dies zeigen die beiden Tabellen [22.5] und [22.6].

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Tabelle 22.5: Addition modulo 8.

Die Multiplikation modulo 8 liefert die Tabelle [22.6].

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Tabelle 22.6: Multiplikation modulo 8.

Man erkennt, dass in der Multiplikationstabelle [22.6] die nichtverschwindenden Zahlen nicht gleichmäßig auftreten. Die 3 tritt beispielweise nur viermal auf, die Zahl 4 jedoch zwölfmal.

Ganze Zahl	1	2	3	4	5	6	7
Auftreten in $\mathbb{Z}_8$	4	8	4	12	4	8	4
Auftreten in $\mathtt{GF}(2^3)$	7	7	7	7	7	7	7

Wir werden sehen, dass es endliche Körper der Form $\mathtt{GF}(2^n)$ gibt. Insbesondere existiert ein endlicher Körper mit $n = 3$, der acht Elemente enthält. Die Verknüpfungstabellen sind in [22.7] und [22.8] gezeigt, wobei wir momentan die Frage ausser Acht lassen, wie diese Tabellen entstehen.

Anmerkungen:

(a) Die Additions– und Multiplikationstabelle ist jeweils symmetrisch zur Hauptdiagonale, dies ist konform zur Kommutativität der Additions– bzw. Multiplikationsoperation.

		000	001	010	011	100	101	110	111
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
000	**0**	0	1	2	3	4	5	6	7
001	**1**	1	0	3	2	5	4	7	6
010	**2**	2	3	0	1	6	7	4	5
011	**3**	3	2	1	0	7	6	5	4
100	**4**	4	5	6	7	0	1	2	3
101	**5**	5	4	7	6	1	0	3	2
110	**6**	6	7	4	5	2	3	0	1
111	**7**	7	6	5	4	3	2	1	0

Tabelle 22.7: Addition in $\mathtt{GF}\left(2^3\right)$.

		000	001	010	011	100	101	110	111
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
000	**0**	0	0	0	0	0	0	0	0
001	**1**	0	1	2	3	4	5	6	7
010	**2**	0	2	4	6	3	1	7	5
011	**3**	0	3	6	5	7	4	1	2
100	**4**	0	4	3	7	6	2	5	1
101	**5**	0	5	1	4	2	7	3	6
110	**6**	0	6	7	1	5	3	2	4
111	**7**	0	7	5	2	1	6	4	3

Tabelle 22.8: Multiplikation in $\mathtt{GF}\left(2^3\right)$.

(b) Alle Elemente aus $\mathtt{GF}\left(2^3\right)$, die nicht 0 sind, haben ein multiplikatives Inverses.

(c) Die Tabellen [22.7] und [22.8] führen auf die algebraische Struktur eines Körpers, dies ist der endliche Körper $\mathtt{GF}\left(2^3\right)$.

Intuitiv ist es klar, dass ein kryptographischer Algorithmus, der die ganzen Zahlen ungleichmäßig abbildet, kryptographisch schwächer ist gegenüber einem Verfahren, das die Zahlen gleichverteilt abbildet. Das heißt, dass kryptoanalytische Verfahren diesen Sachverhalt offenlegen und damit einen Angriffspunkt haben. Daher sind endliche Körper der Form $\mathtt{GF}\left(2^n\right)$ attraktiv für kryptographische Algorithmen.

Zusammenfassend kann man daher sagen, dass wir eine Menge suchen,[9] die aus 2^n Elementen besteht und auf der wir Addition und Multiplikation auf eine Weise definieren können, dass die algebraische Struktur eines Körpers entsteht. Wir können jedem Element dieser Menge eine eindeutige Zahl im Bereich 0 bis

[9] Diese Menge muss keine Menge von Zahlen sein.

$2^n - 1$ zuordnen. Man beachte, dass wir *keine* modulare Arithmetik verwenden können, da dieses Konstrukt nicht auf eine Körperstruktur führt. Wir werden sehen, dass Polynom Arithmetik die geeigneten Werkzeuge liefert.

22.3.2 Modulare Polynom Arithmetik

Wir betrachten die Menge S aller Polynome vom Grad $n-1$ oder kleiner auf dem Körper $\mathbb{Z}_p$. Jedes Polynom hat die Form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n-1} a_i x^i,$$

mit $a_i \in \mathbb{Z}_p$ für alle $i = 0, 1, 2, \ldots, n-1$. Die Menge S enthält insgesamt p^n Polynome.

Beispiel [22.71]

(a) Für $p = 3, n = 2$ sind die $3^2 = 9$ Polynome:

$$0, 1, 2, x, x+1, x+1, 2x, 2x+1, 2x+2.$$

(b) Für $p = 2, n = 3$ haben wir $2^3 = 8$ Polynome, diese sind

$$0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1.$$

Mit einer geeigneten Definition der arithmetischen Operationen wird auf jeder dieser Mengen eine Körperstruktur induziert. Diese Definition besteht aus folgenden Elementen.

1. Die Arithmetik ist identisch mit der gewöhnlichen Arithmetik von Polynomen, es gelten aber die folgenden beiden Modifikationen.
2. Die Arithmetik der Koeffizienten ist modulo p zu nehmen. Die Koeffizienten der Polynome unterliegen also den Regeln der Arithmetik des endlichen Körpers $\mathbb{Z}_p$.
3. Falls die Multiplikation zweier Polynome auf ein Polynom führt mit Grad größer $n-1$, dann ist dieses Polynom zu reduzieren modulo einem irreduziblen Polynom $m(x)$ vom Grad n. Das bedeutet, wir dividieren das Produktpolynom durch $m(x)$ und behalten den Rest. Für ein Polynom $f(x)$ schreibt man dann den Rest als

$$r(x) = f(x) \bmod m(x).$$

Beispiel [22.72]

Der AES arbeitet mit der Arithmetik auf dem Körper $\mathtt{GF}\left(2^8\right)$ mit dem irreduziblen Polynom

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{22.7}$$

Betrachte die beiden Polynome

$$\begin{aligned} f(x) &= x^6 + x^4 + x^2 + x + 1, \\ g(x) &= x^7 + x + 1. \end{aligned}$$

Dann ist die Summe dieser beiden Polynome:

$$\begin{aligned} f(x) + g(x) &= \left(x^6 + x^4 + x^2 + x + 1\right) + \left(x^7 + x + 1\right) \\ &= x^7 + x^6 + x^4 + x^2. \end{aligned}$$

Das Produkt ist:

$$\begin{aligned} f(x) \times g(x) &= \left(x^6 + x^4 + x^2 + x + 1\right) \times \left(x^7 + x + 1\right) \\ &= x^{13} + x^7 + x^6 \\ &\quad + x^{11} + x^5 + x^4 \\ &\quad + x^9 + x^3 + x^2 \\ &\quad + x^8 + x^2 + x \\ &\quad + x^7 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1. \end{aligned}$$

Der Grad dieses Produktpolynoms ist größer 8, daher führen wir die Reduktion modulo $m(x)$ durch:

$$\begin{array}{l} (x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) : (x^8 + x^4 + x^3 + x + 1) = x^5 + x^3 \\ \underline{x^{13} \quad + x^9 + x^8 + x^6 + x^5} \\ \quad x^{11} \qquad\qquad\qquad\qquad + x^4 + x^3 + 1 \\ \quad \underline{x^{11} \qquad + x^7 + x^6 \qquad + x^4 + x^3} \\ \qquad\qquad\quad x^7 + x^6 + 1 \end{array}$$

Damit ist

$$f(x) \cdot g(x) \bmod m(x) = x^7 + x^6 + 1.$$

22.4 Der endliche Körper $GF(2^4)$

Um zu sehen, was es mit GALOIS Feldern auf sich hat, untersuchen wir das Feld $\texttt{GF}(2^4)$ im Detail. Dies ist als Spielmodell zu sehen, um die grundlegenden Konzepte der endlichen Körper kennen zu lernen.

Wir wollen nun explizit den Körper $\texttt{GF}(2^4)$ konstruieren. Dieses Beispiel entspricht also dem Fall $p = 2$ und $m = 4$ im allgemeinen Fall $\texttt{GF}(p^m)$.

Der Körper $\texttt{GF}(2^4)$ hat zunächst 2^4 Elemente, die man als 4-Tupel – genannt Vekoren – schreiben kann. Diese Elemente sind:

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

Diese 4-Tuples kann man auch als Binärzahlen von 0 bis 15 interpretieren, wobei wir die Notation anwenden, dass das kleinstwertige Bit ganz rechts steht (little endian).

Um dieser Menge eine algebraische Struktur zu geben, müssen Additions- und Multiplikationsoperationen auf dieser Menge definiert werden.

Addition und Subtraktion werden Bit-weise modulo 2 durchgeführt, was der Bit-weisen `XOR` Verknüpfung entspricht.

Wir definieren die Addition auf $\texttt{GF}(2^4)$ wie folgt:

Sei

$$a = \sum_{k=0}^{3} a_k 2^k \in \texttt{GF}(2^4) \text{ und } b = \sum_{l=0}^{3} b_l 2^l \in \texttt{GF}(2^4) \text{ mit } a_i, b_i \in \mathbb{Z}_2$$

Dann ist

$$\begin{aligned} \oplus \ : \texttt{GF}(2^4) \times \texttt{GF}(2^4) &\longrightarrow \texttt{GF}(2^4) \\ (a, b) &\longmapsto a \oplus b = \sum_{k=0}^{3} \big[(a_k + b_k) \bmod 2\big] \cdot 2^k. \end{aligned}$$

Addieren wir auf diese Weise die Vektoren $\texttt{0011} \in GF(2^4)$ und $\texttt{1111} \in GF(2^4)$, denn erhalten wir:

$$\begin{aligned} a &= \sum_{k=0}^{3} a_i 2^i = 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ b &= \sum_{k=0}^{3} b_i 2^i = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0. \end{aligned}$$

$\oplus$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Tabelle 22.9: Additionstabelle

Dann ist die Summe:

$$\begin{aligned} a \oplus b &= (0+1) \bmod 2 \ \cdot 2^3 + (0+1) \bmod 2 \ \cdot 2^2 \\ &\quad + (1+1) \bmod 2 \ \cdot 2^1 + (1+1) \bmod 2 \ \cdot 2^0 \\ &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0. \end{aligned}$$

Setzt man

$$a \oplus b = c = \sum_{i=0}^{3} c_i \cdot 2^i$$

dann erhält man durch Vergleich der beiden Summen:

$$c_i = (a_i + b_i) \bmod 2$$

Dies ist eine ungewohnte Art und Weise, Zahlen zu addieren, die nicht mit der üblichen Addition binärer Zahlen korrespondiert, da keine Überträge in die nächsten Stelle berücksichtigt werden.

Mit dieser Additionsoperation ist die Subtraktion identisch mit der Addition, i.e. jedes Element ist sein eigenes Negatives. Diesen Sachverhalt erkennt man leicht anhand der Additionstabelle des Körpers $\mathtt{GF}(2^4)$, die untenstehend aufgeführt ist:

Aus dieser Tabelle erkennt man, dass die Diagonalelemente den Wert 0 haben, dies korrespondiert mit $x \oplus x = 0$, was genau bedeutet, dass

$$x = -x \quad \forall\, x \in \mathtt{GF}(2^4).$$

Anzumerken ist, dass diese Eigenschaft seinen Ursprung in der grundlegenden Eigenschaft der `XOR`-Operation hat.

Behauptung:

Die Abbildung

$$\oplus \ : \ \mathtt{GF}(2^4) \times \mathtt{GF}(2^4) \ \longrightarrow \ \mathtt{GF}(2^4)$$

ist kommutativ und assoziativ, daher ist das Paar $(\mathtt{GF}(2^4), \oplus)$ eine ABELsche Gruppe..

Beweis:

Die Kommutativität – i.e. die Eigenschaft $a \oplus b = b \oplus a$ – ist trivial, da

$$(a_i + b_i) \bmod 2 = (b_i + a_i) \bmod 2$$

Eine andere Weise, dies zu sehen, besteht darin, dass die Additionstabelle symmetrisch zur Diagonalen ist.

Um das Assoziativitätsgesetz zu verifizieren, müssen wir zeigen:

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c.$$

Aus der Additionstabelle liest man beispielsweise ab:

$$(7 \oplus 3) \oplus 4 = 4 \oplus 4 = 0$$

und

$$7 \oplus (3 \oplus 4) = 7 \oplus 7 = 0.$$

Ein weiteres Beispiel:

$$(6 \oplus 11) \oplus 7 = 13 \oplus 7 = 10$$

und

$$6 \oplus (11 \oplus 7) = 6 \oplus 12 = 10$$

Um einen allgemeingültigen Beweis des Assoziativitätsgesetz zu geben, betrachten wir drei Elemente aus $\mathtt{GF}(2^4)$:

$$a = \sum_{i=0}^{3} a_i \cdot 2^i$$
$$b = \sum_{i=0}^{3} b_i \cdot 2^i$$
$$c = \sum_{i=0}^{3} c_i \cdot 2^i$$

Gemäß Definition haben wir:

$$a \oplus b = \sum_{k=0}^{3} [(a_k + b_k) \bmod 2] \cdot 2^k = \sum_{k=0}^{3} d_k \cdot 2^k$$

mit

$$d_k = (a_k + b_k) \bmod 2.$$

Dann ist:

$$\begin{aligned}(a \oplus b) \oplus c &= \left(\sum_{i=0}^{3}[(a_i + b_i) \bmod 2] \cdot 2^i\right) \oplus c \\ &= \left(\sum_{i=0}^{3} d_i \cdot 2^i\right) \oplus c \\ &= \sum_{i=0}^{3}[(d_i + c_i) \bmod 2] \cdot 2^i \\ &= \sum_{k=0}\left[\left((a_k + b_k) \bmod 2 + c_k\right) \bmod 2\right] \cdot 2^k.\end{aligned}$$

Verwendet man zweimal die grundlegende Eigenschaft der modularen Addition:

$$[(a(\bmod n)) + (b(\bmod n))](\bmod n) \equiv (a + b)(\bmod n)$$

dann können wir schreiben:

$$\begin{aligned}(a \oplus b) \oplus c &= \sum_{k=0}\left[\left(a_k + b_k + c_k\right) \bmod 2\right] \cdot 2^k \\ &= \sum_{k=0}\left[\left(a_k + (b_k + c_k) \bmod 2\right) \bmod 2\right] \cdot 2^k \\ &= a \oplus \left(\sum_{k=0}^{3}[(b_k + c_k) \bmod 2] \cdot 2^k\right) \\ &= a \oplus (b \oplus c).\end{aligned}$$

Damit ist gezeigt, dass die Menge $\mathtt{GF}(2^4)$ mit der Operation $\oplus$ eine ABELsche Gruppe ist.

22.4.1 Polynome über $GF(2^4)$

Die folgende Liste stellt eine Abbildung der Vektoren aus $\texttt{GF}(2^4)$ in die Menge der Polynome über $\texttt{GF}(2^4)$ dar:

Bit String	Polynom	Bezeichnung
0000	0	p_0
0001	1	p_1
0010	x	p_2
0011	$x+1$	p_3
0100	x^2	p_4
0101	x^2+1	p_5
0110	x^2+x	p_6
0111	x^2+x+1	p_7
1000	x^3	p_8
1001	x^3+1	p_9
1010	x^3+x	p_{10}
1011	x^3+x+1	p_{11}
1100	x^3+x^2	p_{12}
1101	x^3+x^2+1	p_{13}
1110	x^3+x^2+x	p_{14}
1111	x^3+x^2+x+1	p_{15}

Die obige Tabelle zeigt die Korrespondenz zwischen den Polynomen über $\mathbb{Z}_2$ und den sechzehn 4-Tupeln aus $\texttt{GF}(2^4)$.

Die Additionsoperation für die Polynome bedeutet hier Addition modulo 2.

Im allgemeinen Fall, $\texttt{GF}(p^m)$ gibt es genau p^m solcher Polynome, i.e. die Anzahl der Polynome ist identlich mit der Ordnung des Feldes. Jedes Polynom entspricht genau einem Element des Körpers. Daher entspricht im Feld $\texttt{GF}(2^4)$ dem Element 0011 das Polynom

$$0+0+x^1+x^0 = x+1$$

und das Element 1010 wird durch das Polynom

$$x^3+0+x^1+0 = x^3+x$$

dargestellt.

Die **Multiplikation** im GALOIS Feld $\texttt{GF}(p^m)$ ist definiert als *Multiplikation von Polynomen modulo einem gegebenen irreduziblen Polynom $p(x)$* über $\texttt{GF}(p^m)$ vom Grad m.

Im Beispiel [22.65] haben wir abgeleitet, dass es in $\texttt{GF}(2^4)$ die drei irreduziblen Polynome 4. Grades

$$\begin{aligned}\pi_1(x) &= p_9(x) = x^4+x+1\\ \pi_2(x) &= p_{11}(x) = x^4+x^3+1\\ \pi_3(x) &= p_{15}(x) = x^4+x^3+x^2+x+1\end{aligned}$$

gibt[10]. Wir definieren die Multiplikation auf $\mathtt{GF}(2^4)$ folgendermaßen.

Definition

Seien $g(x)$ und $h(x)$ zwei Polynome, die Elemente aus $\mathtt{GF}(2^4)$ repräsentieren. Dann ist:

$$\otimes \; : \; \mathtt{GF}(2^4)/(\pi_1) \longrightarrow \mathtt{GF}(2^4)/(\pi_1)$$

mit:

$$g(x) \otimes h(x) = (g(x) \cdot h(x)) \bmod \pi_1(x)$$

Beispiel [22.73]
Wir betrachten die beiden Polynome p_9, p_{11} auf $\mathtt{GF}(2^4)$:

$$p_{11}(x) = x^3 + x + 1$$
$$p_9(x) = x^3 + 1$$

Multiplizieren wir diese beiden Polynome, erhalten wir:

$$p_{11}(x) \cdot p_9(x) = x^6 + x^4 + x + 1.$$

Nun wird dieses Produktpolynom durch das irreduzible Polynom $\pi_2(x)$ dividiert:

$$\begin{array}{l} x^6 + x^4 + x + 1 \; : \; x^4 + x^3 + 1 \; = \; x^2 + x + 1 \\ \underline{x^6 + x^5 + x^2} \\ \quad x^5 + x^4 + x^2 + x + 1 \\ \quad \underline{x^5 + x^4 \qquad + x} \\ \qquad x^2 + 1 \end{array}$$

Wir erhalten damit:

$$(x^3 + x + 1) \otimes (x^3 + 1) = x^2 + 1$$

Beispiel [22.74] Wir greifen nochmal das Beispiel [22.65] auf. Der Restklas-

[10] Für die irreduziblen Polynome führen wir zur Vermeidung von Mißverständnissen eine andere Notation ein und bezeichnen sie mit π_1, π_2 respektive π_3.

senring $\mathbb{F}_{2^4}/(\pi_2)$ besteht aus den $p^n = 2^4 = 16$ Klassen:

$$\begin{aligned}\mathbb{F}_{2^4}/(\pi_2) = \Big\{&[0],[1],\\ &[x],[x+1],\\ &[x^2],[x^2+1],[x^2+x],[x^2+x+1],\\ &[x^3],[x^3+1],[x^3+x],[x^3+x^2],\\ &[x^3+x+1],[x^3+x^2+1],[x^3+x^2+x],\\ &[x^3+x^2+x+1]\Big\}\\ = \{&[p_0],[p_1],\ldots,[p_{15}]\}.\end{aligned}$$

Die in der Tabelle [22.10] dargestellte Addition auf diesem Ring ist eine Abbildung:

$$\oplus : \mathbb{F}_{2^4}/(\pi_2) \times \mathbb{F}_{2^4}/(\pi_2) \longrightarrow \mathbb{F}_{2^4}/(\pi_2)$$

Explizit erhalten wir:

1. [0] und [1]:

$$[0] \oplus [p] = [p] \; \forall [p] \in \mathbb{F}_{2^4}/(p_9)$$

i.e. das Element [0] ist das neutrale Element der Addition.

$$\begin{aligned}[1] \oplus [1] &= [0];\\ [1] \oplus [x] &= [x+1];\\ [1] \oplus [x+1] &= [x];\\ [1] \oplus [x^2] &= [x^2+1];\\ [1] \oplus [x^2+1] &= [x^2];\\ [1] \oplus [x^2+x] &= [x^2+x+1];\\ [1] \oplus [x^2+x+1] &= [x^2+x];\end{aligned}$$

$$\begin{aligned}[1] \oplus [x^3] &= [x^3+1];\\ [1] \oplus [x^3+1] &= [x^3];\\ [1] \oplus [x^3+x] &= [x^3+x+1];\\ [1] \oplus [x^3+x+1] &= [x^3+x];\\ [1] \oplus [x^3+x^2] &= [x^3+x^2+1];\\ [1] \oplus [x^3+x^2+1] &= [x^3+x^2];\\ [1] \oplus [x^3+x^2+x] &= [x^3+x^2+x+1]\\ [1] \oplus [x^3+x^2+x+1] &= [x^3+x^2+x]\end{aligned}$$

2. $[x]$:

$$\begin{aligned}
[x] \oplus [x] &= [0];\\
[x] \oplus [x+1] &= [1];\\
[x] \oplus [x^2] &= [x^2+x];\\
[x] \oplus [x^2+1] &= [x^2+x+1];\\
[x] \oplus [x^2+x] &= [x^2];\\
[x] \oplus [x^2+x+1] &= [x^2+1];
\end{aligned}$$

$$\begin{aligned}
[x] \oplus [x^3] &= [x^3+x];\\
[x] \oplus [x^3+1] &= [x^3+x+1];\\
[x] \oplus [x^3+x] &= [x^3];\\
[x] \oplus [x^3+x+1] &= [x^3+1];\\
[x] \oplus [x^3+x^2] &= [x^3+x^2+x];\\
[x] \oplus [x^3+x^2+1] &= [x^3+x^2+x+1];\\
[x] \oplus [x^3+x^2+x] &= [x^3+x^2]\\
[x] \oplus [x^3+x^2+x+1] &= [x^3+x^2+1]
\end{aligned}$$

3. $[x+1]$:

$$\begin{aligned}
[x+1] \oplus [x+1] &= [0];\\
[x+1] \oplus [x^2] &= [x^2+x+1];\\
[x+1] \oplus [x^2+1] &= [x^2+x];\\
[x+1] \oplus [x^2+x] &= [x^2+1];\\
[x+1] \oplus [x^2+x+1] &= [x^2];
\end{aligned}$$

$$\begin{aligned}
[x+1] \oplus [x^3] &= [x^3+x+1];\\
[x+1] \oplus [x^3+1] &= [x^3+x];\\
[x+1] \oplus [x^3+x] &= [x^3+1];\\
[x+1] \oplus [x^3+x+1] &= [x^3];\\
[x+1] \oplus [x^3+x^2] &= [x^3+x^2+x+1];\\
[x+1] \oplus [x^3+x^2+1] &= [x^3+x^2+x];\\
[x+1] \oplus [x^3+x^2+x] &= [x^3+x^2+1]\\
[x+1] \oplus [x^3+x^2+x+1] &= [x^3+x^2]
\end{aligned}$$

4. $[x^2]$:

$$\begin{aligned}
[x^2] \oplus [x^2] &= [0];\\
[x^2] \oplus [x^2+1] &= [1];\\
[x^2] \oplus [x^2+x] &= [x];\\
[x^2] \oplus [x^2+x+1] &= [x+1];
\end{aligned}$$

$$
\begin{aligned}
[x^2] \oplus [x^3] &= [x^3 + x^2];\\
[x^2] \oplus [x^3 + 1] &= [x^3 + x^2 + 1];\\
[x^2] \oplus [x^3 + x] &= [x^3 + x^2 + x];\\
[x^2] \oplus [x^3 + x + 1] &= [x^3 + x^2 + x + 1];\\
[x^2] \oplus [x^3 + x^2] &= [x^3];\\
[x^2] \oplus [x^3 + x^2 + 1] &= [x^3 + 1];\\
[x^2] \oplus [x^3 + x^2 + x] &= [x^3 + x]\\
[x^2] \oplus [x^3 + x^2 + x + 1] &= [x^3 + x + 1]
\end{aligned}
$$

5. $[x^2 + 1]$:

$$
\begin{aligned}
[x^2 + 1] \oplus [x^2 + 1] &= [0];\\
[x^2 + 1] \oplus [x^2 + x] &= [x + 1];\\
[x^2 + 1] \oplus [x^2 + x + 1] &= [x];
\end{aligned}
$$

$$
\begin{aligned}
[x^2 + 1] \oplus [x^3] &= [x^3 + x^2 + 1];\\
[x^2 + 1] \oplus [x^3 + 1] &= [x^3 + x^2];\\
[x^2 + 1] \oplus [x^3 + x] &= [x^3 + x^2 + x + 1];\\
[x^2 + 1] \oplus [x^3 + x + 1] &= [x^3 + x^2 + x];\\
[x^2 + 1] \oplus [x^3 + x^2] &= [x^3 + 1];\\
[x^2 + 1] \oplus [x^3 + x^2 + 1] &= [x^3];\\
[x^2 + 1] \oplus [x^3 + x^2 + x] &= [x^3 + x + 1]\\
[x^2 + 1] \oplus [x^3 + x^2 + x + 1] &= [x^3 + x]
\end{aligned}
$$

6. $[x^2 + x]$:

$$
\begin{aligned}
[x^2 + x] \oplus [x^2 + x] &= [0];\\
[x^2 + x] \oplus [x^2 + x + 1] &= [1];\\
[x^2 + x] \oplus [x^3] &= [x^3 + x^2 + x];\\
[x^2 + x] \oplus [x^3 + 1] &= [x^3 + x^2 + x + 1];\\
[x^2 + x] \oplus [x^3 + x] &= [x^3 + x^2];\\
[x^2 + x] \oplus [x^3 + x + 1] &= [x^3 + x^2 + 1];\\
[x^2 + x] \oplus [x^3 + x^2] &= [x^3 + x];\\
[x^2 + x] \oplus [x^3 + x^2 + 1] &= [x^3 + x^2];\\
[x^2 + x] \oplus [x^3 + x^2 + x] &= [x^3]\\
[x^2 + x] \oplus [x^3 + x^2 + x + 1] &= [x^3 + 1]
\end{aligned}
$$

7. $[x^2+x+1]$:

$$\begin{aligned}
[x^2+x+1] \oplus [x^2+x+1] &= [0];\\
[x^2+x+1] \oplus [x^3] &= [x^3+x^2+x+1];\\
[x^2+x+1] \oplus [x^3+1] &= [x^3+x^2+x];\\
[x^2+x+1] \oplus [x^3+x] &= [x^3+x^2+1];\\
[x^2+x+1] \oplus [x^3+x+1] &= [x^3+x^2];\\
[x^2+x+1] \oplus [x^3+x^2] &= [x^3+x+1];\\
[x^2+x+1] \oplus [x^3+x^2+1] &= [x^3+x];\\
[x^2+x+1] \oplus [x^3+x^2+x] &= [x^3+1]\\
[x^2+x+1] \oplus [x^3+x^2+x+1] &= [x^3]
\end{aligned}$$

8. $[x^3]$:

$$\begin{aligned}
[x^3] \oplus [x^3] &= [0];\\
[x^3] \oplus [x^3+1] &= [1];\\
[x^3] \oplus [x^3+x] &= [x];\\
[x^3] \oplus [x^3+x+1] &= [x+1];\\
[x^3] \oplus [x^3+x^2] &= [x^2];\\
[x^3] \oplus [x^3+x^2+1] &= [x^2+1];\\
[x^3] \oplus [x^3+x^2+x] &= [x^2+x]\\
[x^3] \oplus [x^3+x^2+x+1] &= [x^2+x+1]
\end{aligned}$$

9. $[x^3+1]$:

$$\begin{aligned}
[x^3+1] \oplus [x^3+1] &= [0];\\
[x^3+1] \oplus [x^3+x] &= [x+1];\\
[x^3+1] \oplus [x^3+x+1] &= [x];\\
[x^3+1] \oplus [x^3+x^2] &= [x^2+1];\\
[x^3+1] \oplus [x^3+x^2+1] &= [x^2];\\
[x^3+1] \oplus [x^3+x^2+x] &= [x^2+x+1]\\
[x^3+1] \oplus [x^3+x^2+x+1] &= [x^2+x]
\end{aligned}$$

10. $[x^3+x]$:

$$\begin{aligned}
[x^3+x] \oplus [x^3+x] &= [0];\\
[x^3+x] \oplus [x^3+x+1] &= [1];\\
[x^3+x] \oplus [x^3+x^2] &= [x^2+x];\\
[x^3+x] \oplus [x^3+x^2+1] &= [x^2+x+1];\\
[x^3+x] \oplus [x^3+x^2+x] &= [x^2]\\
[x^3+x] \oplus [x^3+x^2+x+1] &= [x^2+1]
\end{aligned}$$

11. $[x^3+x+1]$:

$$\begin{aligned}
[x^3+x+1] \oplus [x^3+x+1] &= [0];\\
[x^3+x+1] \oplus [x^3+x^2] &= [x^2+x+1];\\
[x^3+x+1] \oplus [x^3+x^2+1] &= [x^2+x];\\
[x^3+x+1] \oplus [x^3+x^2+x] &= [x^2+1]\\
[x^3+x+1] \oplus [x^3+x^2+x+1] &= [x^2]
\end{aligned}$$

12. $[x^3+x^2]$:

$$\begin{aligned}
[x^3+x^2] \oplus [x^3+x^2] &= [0];\\
[x^3+x^2] \oplus [x^3+x^2+1] &= [1];\\
[x^3+x^2] \oplus [x^3+x^2+x] &= [x]\\
[x^3+x^2] \oplus [x^3+x^2+x+1] &= [x+1]
\end{aligned}$$

13. $[x^3+x^2+1]$, $[x^3+x^2+x]$, $[x^3+x^2+x+1]$:

$$\begin{aligned}
[x^3+x^2+1] \oplus [x^3+x^2+1] &= [0];\\
[x^3+x^2+1] \oplus [x^3+x^2+x] &= [x+1]\\
[x^3+x^2+1] \oplus [x^3+x^2+x+1] &= [x]\\
[x^3+x^2+x] \oplus [x^3+x^2+x] &= [0]\\
[x^3+x^2+x] \oplus [x^3+x^2+x+1] &= [1]\\
[x^3+x^2+x+1] \oplus [x^3+x^2+x+1] &= [0]
\end{aligned}$$

In der Tabelle [22.10] ist die komplette Additionstabelle der Polynome 3. Grades über $\mathbb{F}_2$ dargestellt.

Die Multiplikation auf diesem Ring ist eine Abbildung:

$$\otimes : \ \mathbb{F}_{2^4}/(\pi_2) \times \mathbb{F}_{2^4}/(\pi_2) \ \longrightarrow \ \mathbb{F}_{2^4}/(\pi_2),$$

wobei das Produktpolynom reduziert wird modulo π_2, i.e.

$$[q_i] \otimes [q_j] = [q_i \cdot q_j \bmod \pi_2] \quad \forall\, q_i, q_j \in \mathbb{F}_{2^4}/(\pi_2).$$

Explizit erhalten wir für die Multiplikation:

1. [0] und [1]:

$$[0] \otimes [p] = [0], \quad \forall\, [p] \in \mathbb{F}_{2^4}/(\pi_2)$$

$$[1] \otimes [p] = [p], \quad \forall\, [p] \in \mathbb{F}_{2^4}/(\pi_2)$$

   i.e. das Element [1] ist das neutrale Element der Multiplikation.

$\oplus$	$[p_0]$	$[p_1]$	$[p_2]$	$[p_3]$	$[p_4]$	$[p_5]$	$[p_6]$	$[p_7]$	$[p_8]$	$[p_9]$	$[p_{10}]$	$[p_{11}]$	$[p_{12}]$	$[p_{13}]$	$[p_{14}]$	$[p_{15}]$
$[p_0]$	$[p_0]$	$[p_1]$	$[p_2]$	$[p_3]$	$[p_4]$	$[p_5]$	$[p_6]$	$[p_7]$	$[p_8]$	$[p_9]$	$[p_{10}]$	$[p_{11}]$	$[p_{12}]$	$[p_{13}]$	$[p_{14}]$	$[p_{15}]$
$[p_1]$	$[p_1]$	$[p_0]$	$[p_3]$	$[p_2]$	$[p_5]$	$[p_4]$	$[p_7]$	$[p_6]$	$[p_9]$	$[p_8]$	$[p_{11}]$	$[p_{10}]$	$[p_{13}]$	$[p_{12}]$	$[p_{15}]$	$[p_{14}]$
$[p_2]$	$[p_2]$	$[p_3]$	$[p_0]$	$[p_1]$	$[p_6]$	$[p_7]$	$[p_4]$	$[p_5]$	$[p_{10}]$	$[p_{11}]$	$[p_8]$	$[p_9]$	$[p_{14}]$	$[p_{15}]$	$[p_{12}]$	$[p_{13}]$
$[p_3]$	$[p_3]$	$[p_2]$	$[p_1]$	$[p_0]$	$[p_7]$	$[p_6]$	$[p_5]$	$[p_4]$	$[p_{11}]$	$[p_{10}]$	$[p_9]$	$[p_8]$	$[p_{15}]$	$[p_{14}]$	$[p_{13}]$	$[p_{12}]$
$[p_4]$	$[p_4]$	$[p_5]$	$[p_6]$	$[p_7]$	$[p_0]$	$[p_1]$	$[p_2]$	$[p_3]$	$[p_{12}]$	$[p_{13}]$	$[p_{14}]$	$[p_{15}]$	$[p_8]$	$[p_9]$	$[p_{10}]$	$[p_{11}]$
$[p_5]$	$[p_5]$	$[p_4]$	$[p_7]$	$[p_6]$	$[p_1]$	$[p_0]$	$[p_3]$	$[p_2]$	$[p_{13}]$	$[p_{12}]$	$[p_{15}]$	$[p_{14}]$	$[p_9]$	$[p_8]$	$[p_{11}]$	$[p_{10}]$
$[p_6]$	$[p_6]$	$[p_7]$	$[p_4]$	$[p_5]$	$[p_2]$	$[p_3]$	$[p_0]$	$[p_1]$	$[p_{14}]$	$[p_{15}]$	$[p_{12}]$	$[p_{13}]$	$[p_{10}]$	$[p_{11}]$	$[p_8]$	$[p_9]$
$[p_7]$	$[p_7]$	$[p_6]$	$[p_5]$	$[p_4]$	$[p_3]$	$[p_2]$	$[p_1]$	$[p_0]$	$[p_{15}]$	$[p_{14}]$	$[p_{13}]$	$[p_{12}]$	$[p_{11}]$	$[p_{10}]$	$[p_9]$	$[p_8]$
$[p_8]$	$[p_8]$	$[p_9]$	$[p_{10}]$	$[p_{11}]$	$[p_{12}]$	$[p_{13}]$	$[p_{14}]$	$[p_{15}]$	$[p_0]$	$[p_1]$	$[p_2]$	$[p_3]$	$[p_4]$	$[p_5]$	$[p_6]$	$[p_7]$
$[p_9]$	$[p_9]$	$[p_8]$	$[p_{11}]$	$[p_{10}]$	$[p_{13}]$	$[p_{12}]$	$[p_{15}]$	$[p_{14}]$	$[p_1]$	$[p_0]$	$[p_3]$	$[p_2]$	$[p_5]$	$[p_4]$	$[p_7]$	$[p_6]$
$[p_{10}]$	$[p_{10}]$	$[p_{11}]$	$[p_8]$	$[p_9]$	$[p_{14}]$	$[p_{15}]$	$[p_{12}]$	$[p_{13}]$	$[p_2]$	$[p_3]$	$[p_0]$	$[p_1]$	$[p_6]$	$[p_7]$	$[p_4]$	$[p_5]$
$[p_{11}]$	$[p_{11}]$	$[p_{10}]$	$[p_9]$	$[p_8]$	$[p_{15}]$	$[p_{14}]$	$[p_{13}]$	$[p_{12}]$	$[p_3]$	$[p_2]$	$[p_1]$	$[p_0]$	$[p_7]$	$[p_6]$	$[p_5]$	$[p_4]$
$[p_{12}]$	$[p_{12}]$	$[p_{13}]$	$[p_{14}]$	$[p_{15}]$	$[p_8]$	$[p_9]$	$[p_{10}]$	$[p_{11}]$	$[p_4]$	$[p_5]$	$[p_6]$	$[p_7]$	$[p_0]$	$[p_1]$	$[p_2]$	$[p_3]$
$[p_{13}]$	$[p_{13}]$	$[p_{12}]$	$[p_{15}]$	$[p_{14}]$	$[p_9]$	$[p_8]$	$[p_{11}]$	$[p_{10}]$	$[p_5]$	$[p_4]$	$[p_7]$	$[p_6]$	$[p_1]$	$[p_0]$	$[p_3]$	$[p_2]$
$[p_{14}]$	$[p_{14}]$	$[p_{15}]$	$[p_{12}]$	$[p_{13}]$	$[p_{10}]$	$[p_{11}]$	$[p_8]$	$[p_9]$	$[p_6]$	$[p_7]$	$[p_4]$	$[p_5]$	$[p_2]$	$[p_3]$	$[p_0]$	$[p_1]$
$[p_{15}]$	$[p_{15}]$	$[p_{14}]$	$[p_{13}]$	$[p_{12}]$	$[p_{11}]$	$[p_{10}]$	$[p_9]$	$[p_8]$	$[p_7]$	$[p_6]$	$[p_5]$	$[p_4]$	$[p_3]$	$[p_2]$	$[p_1]$	$[p_0]$

Tabelle 22.10: Additionstabelle der Polynome 3. Grades über $\mathbb{F}_2$; diese Tabelle hat die gleiche Struktur wie die Tabelle [22.9].

2. $[x]$:

$$\begin{aligned}
[x] \otimes [x] &= [x^2] = [p_4];\\
[x] \otimes [x+1] &= [x^2+x] = [p_6];\\
[x] \otimes [x^2] &= [x^3] = [p_8];\\
[x] \otimes [x^2+1] &= [x^3+x] = [p_{10}];\\
[x] \otimes [x^2+x] &= [x^3+x^2] = [p_{12}];\\
[x] \otimes [x^2+x+1] &= [x^3+x^2+x] = [p_{14}];
\end{aligned}$$

$$\begin{aligned}
[x] \otimes [x^3] &= [x^4] &&= [x^3+1];\\
[x] \otimes [x^3+1] &= [x^4+x] &&= [x^3+x+1];\\
[x] \otimes [x^3+x] &= [x^4+x^2] &&= [x^3+x^2+1];\\
[x] \otimes [x^3+x+1] &= [x^4+x^2+x] &&= [x^3+x^2+x+1];\\
[x] \otimes [x^3+x^2] &= [x^4+x^3] &&= [1];\\
[x] \otimes [x^3+x^2+1] &= [x^4+x^3+x] &&= [x+1];\\
[x] \otimes [x^3+x^2+x] &= [x^4+x^3+x^2] &&= [x^2+1]\\
[x] \otimes [x^3+x^2+x+1] &= [x^4+x^3+x^2+x] &&= [x^2+x+1].
\end{aligned}$$

In der Notation der $[p_i]$ erhalten wir also:

$$\begin{aligned}
[p_2] \otimes [p_2] &= [p_4]\\
[p_2] \otimes [p_3] &= [p_6]\\
[p_2] \otimes [p_4] &= [p_8]\\
[p_2] \otimes [p_5] &= [p_{10}]\\
[p_2] \otimes [p_6] &= [p_{12}]\\
[p_2] \otimes [p_7] &= [p_{14}]\\
[p_2] \otimes [p_8] &= [p_9]\\
[p_2] \otimes [p_9] &= [p_{11}]\\
[p_2] \otimes [p_{10}] &= [p_{13}]\\
[p_2] \otimes [p_{11}] &= [p_{15}]\\
[p_2] \otimes [p_{12}] &= [p_1]\\
[p_2] \otimes [p_{13}] &= [p_3]\\
[p_2] \otimes [p_{14}] &= [p_5]\\
[p_2] \otimes [p_{15}] &= [p_7]
\end{aligned}$$

3. $[x+1]$:

$$\begin{aligned}
[x+1] \otimes [x+1] &= [x^2+1];\\
[x+1] \otimes [x^2] &= [x^3+x^2];\\
[x+1] \otimes [x^2+1] &= [x^3+x^2+x+1];\\
[x+1] \otimes [x^2+x] &= [x^3+x];\\
[x+1] \otimes [x^2+x+1] &= [x^3+1];
\end{aligned}$$

$$
\begin{aligned}
[x+1]\otimes[x^3] &= [x^4] &&= [1];\\
[x+1]\otimes[x^3+1] &= [x^4+x^3+x+1] &&= [x];\\
[x+1]\otimes[x^3+x] &= [x^4+x^3+x^2+x] &&= [x^2+x+1];\\
[x+1]\otimes[x^3+x+1] &= [x^4+x^3+x^2+1] &&= [x^2];\\
[x+1]\otimes[x^3+x^2] &= [x^4+x^2] &&= [x^3+x^2+1];\\
[x+1]\otimes[x^3+x^2+1] &= [x^4+x^2+x+1] &&= [x^3+x^2+x];\\
[x+1]\otimes[x^3+x^2+x] &= [x^4+x] &&= [x^3+x+1]\\
[x+1]\otimes[x^3+x^2+x+1] &= [x^4+1] &&= [x^3].
\end{aligned}
$$

oder:

$$
\begin{aligned}
[p_3]\otimes[p_3] &= [p_5] & [p_3]\otimes[p_9] &= [p_2]\\
[p_3]\otimes[p_4] &= [p_{12}] & [p_3]\otimes[p_{10}] &= [p_7]\\
[p_3]\otimes[p_5] &= [p_{15}] & [p_3]\otimes[p_{11}] &= [p_4]\\
[p_3]\otimes[p_6] &= [p_{10}] & [p_3]\otimes[p_{12}] &= [p_{13}]\\
[p_3]\otimes[p_7] &= [p_9] & [p_3]\otimes[p_{13}] &= [p_{14}]\\
[p_3]\otimes[p_8] &= [p_1] & [p_3]\otimes[p_{14}] &= [p_{11}]\\
& & [p_3]\otimes[p_{15}] &= [p_8]
\end{aligned}
$$

4. $[x^2]$:

$$
\begin{aligned}
[x^2]\otimes[x^2] &= [x^4] &&= [x^3+1];\\
[x^2]\otimes[x^2+1] &= [x^4+x^2] &&= [x^3+x^2+1];\\
[x^2]\otimes[x^2+x] &= [x^4+x^3] &&= [1];\\
[x^2]\otimes[x^2+x+1] &= [x^4+x^3+x^2] &&= [x^2+1];\\
[x^2]\otimes[x^3] &= [x^5] &&= [x^3+x+1];\\
[x^2]\otimes[x^3+1] &= [x^5+x^2] &&= [x^3+x^2+x+1];\\
[x^2]\otimes[x^3+x] &= [x^5+x^3] &&= [x+1];\\
[x^2]\otimes[x^3+x+1] &= [x^5+x^3+x^2] &&= [x^2+x+1];\\
[x^2]\otimes[x^3+x^2] &= [x^5+x^4] &&= [x];\\
[x^2]\otimes[x^3+x^2+1] &= [x^5+x^4+x^2] &&= [x^2+x];\\
[x^2]\otimes[x^3+x^2+x] &= [x^5+x^4+x^3] &&= [x^3+x]\\
[x^2]\otimes[x^3+x^2+x+1] &= [x^5+x^4+x^3+x^2] &&= [x^3+x^2+x].
\end{aligned}
$$

oder:

$$
\begin{aligned}
[p_4]\otimes[p_4] &= [p_9] & [p_4]\otimes[p_{10}] &= [p_3]\\
[p_4]\otimes[p_5] &= [p_{13}] & [p_4]\otimes[p_{11}] &= [p_7]\\
[p_4]\otimes[p_6] &= [p_1] & [p_4]\otimes[p_{12}] &= [p_2]\\
[p_4]\otimes[p_7] &= [p_5] & [p_4]\otimes[p_{13}] &= [p_6]\\
[p_4]\otimes[p_8] &= [p_{11}] & [p_4]\otimes[p_{14}] &= [p_{10}]\\
[p_4]\otimes[p_9] &= [p_{15}] & [p_4]\otimes[p_{15}] &= [p_{14}]
\end{aligned}
$$

5. $[x^2+1]$:

$$
\begin{aligned}
&[x^2+1]\otimes[x^2+1] &&= [x^4+1] &&= [x^3];\\
&[x^2+1]\otimes[x^2+x] &&= [x^4+x^3+x^2+x] &&= [x^2+x+1];\\
&[x^2+1]\otimes[x^2+x+1] &&= [x^4+x^3+x+1] &&= [x];\\
&[x^2+1]\otimes[x^3] &&= [x^5+x^3] &&= [x+1];\\
&[x^2+1]\otimes[x^3+1] &&= [x^5+x^3+x^2+1] &&= [x^2+x];\\
&[x^2+1]\otimes[x^3+x] &&= [x^5+x] &&= [x^3+1];\\
&[x^2+1]\otimes[x^3+x+1] &&= [x^5+x^2+x+1] &&= [x^3+x^2];\\
&[x^2+1]\otimes[x^3+x^2] &&= [x^5+x^4+x^3+x^2] &&= [x^3+x^2+x];\\
&[x^2+1]\otimes[x^3+x^2+1] &&= [x^5+x^4+x^3+1] &&= [x^3+x+1];\\
&[x^2+1]\otimes[x^3+x^2+x] &&= [x^5+x^4+x^2+x] &&= [x^2]\\
&[x^2+1]\otimes[x^3+x^2+x+1] &&= [x^5+x^4+x^3+x^2] &&= [1].
\end{aligned}
$$

oder in der $[p]$-Notation:

$$
\begin{aligned}
&[p_5]\otimes[p_5]=[p_8] && [p_5]\otimes[p_{11}]=[p_{12}]\\
&[p_5]\otimes[p_6]=[p_7] && [p_5]\otimes[p_{12}]=[p_{14}]\\
&[p_5]\otimes[p_7]=[p_2] && [p_5]\otimes[p_{13}]=[p_{11}]\\
&[p_5]\otimes[p_8]=[p_3] && [p_5]\otimes[p_{14}]=[p_4]\\
&[p_5]\otimes[p_9]=[p_6] && [p_5]\otimes[p_{15}]=[p_1]\\
&[p_5]\otimes[p_{10}]=[p_9]
\end{aligned}
$$

6. $[x^2+x]$:

$$
\begin{aligned}
&[x^2+x]\otimes[x^2+x] &&= [x^4+x^2] &&= [x^3+x^2+1];\\
&[x^2+x]\otimes[x^2+x+1] &&= [x^4+x] &&= [x^3+x+1];\\
&[x^2+x]\otimes[x^3] &&= [x^5+x^3] &&= [x];\\
&[x^2+x]\otimes[x^3+1] &&= [x^5+x^4+x^2+x] &&= [x^2];\\
&[x^2+x]\otimes[x^3+x] &&= [x^5+x^4+x^3+x^2] &&= [x^3+x^2+x];\\
&[x^2+x]\otimes[x^3+x+1] &&= [x^5+x^4+x^3+x] &&= [x^3];\\
&[x^2+x]\otimes[x^3+x^2] &&= [x^5+x^3] &&= [x+1];\\
&[x^2+x]\otimes[x^3+x^2+1] &&= [x^5+x^3+x^2+x] &&= [x^2+1];\\
&[x^2+x]\otimes[x^3+x^2+x] &&= [x^5+x^2] &&= [x^3+x^2+x+1]\\
&[x^2+x]\otimes[x^3+x^2+x+1] &&= [x^5+x] &&= [x^3+1].
\end{aligned}
$$

oder in der $[p]$-Notation:

$$
\begin{aligned}
&[p_6]\otimes[p_6]=[p_{13}] && [p_6]\otimes[p_{11}]=[p_8]\\
&[p_6]\otimes[p_7]=[p_{11}] && [p_6]\otimes[p_{12}]=[p_3]\\
&[p_6]\otimes[p_8]=[p_2] && [p_6]\otimes[p_{13}]=[p_5]\\
&[p_6]\otimes[p_9]=[p_4] && [p_6]\otimes[p_{14}]=[p_{15}]\\
&[p_6]\otimes[p_{10}]=[p_{14}] && [p_6]\otimes[p_{15}]=[p_9]
\end{aligned}
$$

7. $[x^2+x+1]$:

$$\begin{aligned}
&[x^2+x+1]\otimes[x^2+x+1] &&= [x^4+x^2+1] &&= [x^3+x^2];\\
&[x^2+x+1]\otimes[x^3] &&= [x^5+x^4+x^3] &&= [x^3+x];\\
&[x^2+x+1]\otimes[x^3+1] &&= [x^5+x^4+x^3+x^2+x+1] &&= [x^3+x^2+1];\\
&[x^2+x+1]\otimes[x^3+x] &&= [x^5+x^4+x^2+x] &&= [x^2];\\
&[x^2+x+1]\otimes[x^3+x+1] &&= [x^5+x^4+1] &&= [x+1];\\
&[x^2+x+1]\otimes[x^3+x^2] &&= [x^5+x^2] &&= [x^3+x^2+x+1];\\
&[x^2+x+1]\otimes[x^3+x^2+1] &&= [x^5+x+1] &&= [x^3];\\
&[x^2+x+1]\otimes[x^3+x^2+x] &&= [x^5+x^3+x] &&= [1]\\
&[x^2+x+1]\otimes[x^3+x^2+x+1] &&= [x^5+x] &&= [x^2+x].
\end{aligned}$$

oder in der $[p]$-Notation:

$$\begin{aligned}
&[p_7]\otimes[p_7]=[p_{12}] && [p_7]\otimes[p_{12}]=[p_{15}]\\
&[p_7]\otimes[p_8]=[p_{10}] && [p_7]\otimes[p_{13}]=[p_8]\\
&[p_7]\otimes[p_9]=[p_{13}] && [p_7]\otimes[p_{14}]=[p_1]\\
&[p_7]\otimes[p_{10}]=[p_4] && [p_7]\otimes[p_{15}]=[p_6]\\
&[p_7]\otimes[p_{11}]=[p_3]
\end{aligned}$$

8. $[x^3]$:

$$\begin{aligned}
&[x^3]\otimes[x^3] &&= [x^6] &&= [x^3+x^2+x+1];\\
&[x^3]\otimes[x^3+1] &&= [x^6+x^3] &&= [x^2+x+1];\\
&[x^3]\otimes[x^3+x] &&= [x^6+x^4] &&= [x^2+x];\\
&[x^3]\otimes[x^3+x+1] &&= [x^6+x^4+x^3] &&= [x^3+x^2+x];\\
&[x^3]\otimes[x^3+x^2] &&= [x^6+x^5] &&= [x^2];\\
&[x^3]\otimes[x^3+x^2+1] &&= [x^6+x^5+x^3] &&= [x^3+x^2];\\
&[x^3]\otimes[x^3+x^2+x] &&= [x^6+x^5+x^4] &&= [x^3+x^2+1]\\
&[x^3]\otimes[x^3+x^2+x+1] &&= [x^6+x^5+x^4+x^3] &&= [x^2+1].
\end{aligned}$$

oder in der $[p]$-Notation:

$$\begin{aligned}
&[p_8]\otimes[p_8]=[p_{15}] && [p_8]\otimes[p_{12}]=[p_4]\\
&[p_8]\otimes[p_9]=[p_7] && [p_8]\otimes[p_{13}]=[p_{12}]\\
&[p_8]\otimes[p_{10}]=[p_6] && [p_8]\otimes[p_{14}]=[p_{13}]\\
&[p_8]\otimes[p_{11}]=[p_{14}] && [p_8]\otimes[p_{15}]=[p_5]
\end{aligned}$$

9. $[x^3+1]$:

$$\begin{aligned}
&[x^3+1]\otimes[x^3+1] &&= [x^6+1] &&= [x^3+x^2+x];\\
&[x^3+1]\otimes[x^3+x] &&= [x^6+x^4+x^3+x] &&= [x^3+x^2];\\
&[x^3+1]\otimes[x^3+x+1] &&= [x^6+x^4+x+1] &&= [x^2+1];\\
&[x^3+1]\otimes[x^3+x^2] &&= [x^6+x^5+x^3+x^2] &&= [x^3];\\
&[x^3+1]\otimes[x^3+x^2+1] &&= [x^6+x^5+x^2+1] &&= [1];\\
&[x^3+1]\otimes[x^3+x^2+x] &&= [x^6+x^5+x^4+x^3+x^2+x] &&= [x]\\
&[x^3+1]\otimes[x^3+x^2+x+1] &&= [x^6+x^5+x^4+x^2+x+1] &&= [x^3+x].
\end{aligned}$$

In der $[p]$-Notation:

$$\begin{aligned}
&[p_9]\otimes[p_9]=[p_{14}] && [p_9]\otimes[p_{13}]=[p_1]\\
&[p_9]\otimes[p_{10}]=[p_{12}] && [p_9]\otimes[p_{14}]=[p_3]\\
&[p_9]\otimes[p_{11}]=[p_5] && [p_9]\otimes[p_{15}]=[p_{10}]\\
&[p_9]\otimes[p_{12}]=[p_8]
\end{aligned}$$

10. $[x^3+x]$:

$$\begin{aligned}
&[x^3+x]\otimes[x^3+x] &&= [x^6+x^2] &&= [x^3+x+1];\\
&[x^3+x]\otimes[x^3+x+1] &&= [x^6+x^3+x^2+x] &&= [1];\\
&[x^3+x]\otimes[x^3+x^2] &&= [x^6+x^5+x^4+x^3] &&= [x^2+1];\\
&[x^3+x]\otimes[x^3+x^2+1] &&= [x^6+x^5+x^4+x] &&= [x^3+x^2+x+1];\\
&[x^3+x]\otimes[x^3+x^2+x] &&= [x^6+x^5+x^3+x^2] &&= [x^3]\\
&[x^3+x]\otimes[x^3+x^2+x+1] &&= [x^6+x^5+x^4+x^3+x^2+x] &&= [x].
\end{aligned}$$

In der $[p]$-Notation:

$$\begin{aligned}
&[p_{10}]\otimes[p_{10}]=[p_{11}] && [p_{10}]\otimes[p_{13}]=[p_{15}]\\
&[p_{10}]\otimes[p_{11}]=[p_1] && [p_{10}]\otimes[p_{14}]=[p_8]\\
&[p_{10}]\otimes[p_{12}]=[p_5] && [p_{10}]\otimes[p_{15}]=[p_2]
\end{aligned}$$

11. $[x^3+x+1]$:

$$\begin{aligned}
&[x^3+x+1]\otimes[x^3+x+1] &&= [x^6+x^2+1] &&= [x^3+x];\\
&[x^3+x+1]\otimes[x^3+x^2] &&= [x^6+x^5+x^4+x^2] &&= [x^3+1];\\
&[x^3+x+1]\otimes[x^3+x^2+1] &&= [x^6+x^5+x^4+x^3+x^2+x+1] &&= [x];\\
&[x^3+x+1]\otimes[x^3+x^2+x] &&= [x^6+x^5+x] &&= [x^2+x]\\
&[x^3+x+1]\otimes[x^3+x^2+x+1] &&= [x^6+x^5+x^3+1] &&= [x^3+x^2+1].
\end{aligned}$$

In der $[p]$-Notation:

$$\begin{aligned}
&[p_{11}]\otimes[p_{11}]=[p_{10}] && [p_{11}]\otimes[p_{14}]=[p_6]\\
&[p_{11}]\otimes[p_{12}]=[p_9] && [p_{11}]\otimes[p_{15}]=[p_{13}]\\
&[p_{11}]\otimes[p_{13}]=[p_2]
\end{aligned}$$

12. $[x^3+x^2], [x^3+x^2+1], [x^3+x^2+x], [x^3+x^2+x+1]$:

$$
\begin{aligned}
&[x^3+x^2]\otimes[x^3+x^2] && = [x^6+x^4] && = [x^2+x];\\
&[x^3+x^2]\otimes[x^3+x^2+1] && = [x^6+x^4+x^3+x^2] && = [x^3+x];\\
&[x^3+x^2]\otimes[x^3+x^2+x] && = [x^6+x^3] && = [x^2+x+1]\\
&[x^3+x^2]\otimes[x^3+x^2+x+1] && = [x^6+x^2] && = [x^3+x+1]\\
&[x^3+x^2+1]\otimes[x^3+x^2+1] && = [x^6+x^4+1] && = [x^2+x+1];\\
&[x^3+x^2+1]\otimes[x^3+x^2+x] && = [x^6+x^3+x] && = [x^2+1]\\
&[x^3+x^2+1]\otimes[x^3+x^2+x+1] && = [x^6+x^3+x+1] && = [x]\\
&[x^3+x^2+x]\otimes[x^3+x^2+x] && = [x^6+x^4+x^2] && = [x]\\
&[x^3+x^2+x]\otimes[x^3+x^2+x+1] && = [x^6+x^4+x^3+x] && = [x^3+x^2]\\
&[x^3+x^2+x+1]\otimes[x^3+x^2+x+1] && = [x^6+x^4+x^2+1] && = [x+1].
\end{aligned}
$$

In der $[p]$-Notation:

$$
\begin{aligned}
&[p_{12}]\otimes[p_{12}] = [p_6] && [p_{13}]\otimes[p_{13}] = [p_7]\\
&[p_{12}]\otimes[p_{13}] = [p_{10}] && [p_{13}]\otimes[p_{14}] = [p_5]\\
&[p_{12}]\otimes[p_{14}] = [p_7] && [p_{13}]\otimes[p_{15}] = [p_2]\\
&[p_{12}]\otimes[p_{15}] = [p_{11}] && [p_{14}]\otimes[p_{14}] = [p_2]\\
& && [p_{14}]\otimes[p_{15}] = [p_{12}]\\
& && [p_{15}]\otimes[p_{15}] = [p_3]
\end{aligned}
$$

In der Tabelle [22.11] ist die komplette Multiplikationstabelle der Polynome 3. Grades über $\mathbb{F}_2$ dargestellt.

Der entscheidende Teil der Multiplikation ist das irreduzible Polynom. Daher ist es sehr sinnvoll, sich etwas näher mit dem Konzept der Irreduzibilität zu befassen.

Definition:

Ein Polynom $\pi(x)$ über einem Feld $\mathcal{F}$ heißt **irreduzibel über** $\mathcal{F}$ genau dann, wenn $\pi(x)$ positiven Grad hat und $\pi = b\cdot c$ mit Polynomen b, c impliziert, dass entweder b oder c ein konstantes Polynom ist.

Mit anderen Worten, ein Polynom mit positiven Grad ist irreduzibel über $\mathcal{F}$, wenn das Polynom nur die triviale Faktorisierung hat[11].

[11] Dies entspricht der Primeigenschaft von Zahlen, i.e. eine Zahl ist Primzahl, wenn die Zahl nur die triviale Faktorisierung $\pm1, \pm p$ hat.

$\otimes$	$[p_0]$	$[p_1]$	$[p_2]$	$[p_3]$	$[p_4]$	$[p_5]$	$[p_6]$	$[p_7]$	$[p_8]$	$[p_9]$	$[p_{10}]$	$[p_{11}]$	$[p_{12}]$	$[p_{13}]$	$[p_{14}]$	$[p_{15}]$
$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$	$[p_0]$
$[p_1]$	$[p_0]$	$[p_1]$	$[p_2]$	$[p_3]$	$[p_4]$	$[p_5]$	$[p_6]$	$[p_7]$	$[p_8]$	$[p_9]$	$[p_{10}]$	$[p_{11}]$	$[p_{12}]$	$[p_{13}]$	$[p_{14}]$	$[p_{15}]$
$[p_2]$	$[p_0]$	$[p_2]$	$[p_4]$	$[p_6]$	$[p_8]$	$[p_{10}]$	$[p_{12}]$	$[p_{14}]$	$[p_9]$	$[p_{11}]$	$[p_{13}]$	$[p_{15}]$	$[p_1]$	$[p_3]$	$[p_5]$	$[p_7]$
$[p_3]$	$[p_0]$	$[p_3]$	$[p_6]$	$[p_5]$	$[p_{12}]$	$[p_{15}]$	$[p_{10}]$	$[p_9]$	$[p_1]$	$[p_2]$	$[p_7]$	$[p_4]$	$[p_{13}]$	$[p_{14}]$	$[p_{11}]$	$[p_8]$
$[p_4]$	$[p_0]$	$[p_4]$	$[p_8]$	$[p_{12}]$	$[p_9]$	$[p_{13}]$	$[p_1]$	$[p_5]$	$[p_{11}]$	$[p_{15}]$	$[p_3]$	$[p_7]$	$[p_2]$	$[p_6]$	$[p_{10}]$	$[p_{14}]$
$[p_5]$	$[p_0]$	$[p_5]$	$[p_{10}]$	$[p_{15}]$	$[p_{13}]$	$[p_8]$	$[p_7]$	$[p_2]$	$[p_3]$	$[p_6]$	$[p_9]$	$[p_{12}]$	$[p_{14}]$	$[p_{11}]$	$[p_4]$	$[p_1]$
$[p_6]$	$[p_0]$	$[p_6]$	$[p_{12}]$	$[p_{10}]$	$[p_1]$	$[p_7]$	$[p_{13}]$	$[p_{11}]$	$[p_2]$	$[p_4]$	$[p_{14}]$	$[p_8]$	$[p_3]$	$[p_5]$	$[p_{15}]$	$[p_9]$
$[p_7]$	$[p_0]$	$[p_7]$	$[p_{14}]$	$[p_9]$	$[p_5]$	$[p_2]$	$[p_{11}]$	$[p_{12}]$	$[p_{10}]$	$[p_{13}]$	$[p_4]$	$[p_3]$	$[p_{15}]$	$[p_8]$	$[p_1]$	$[p_6]$
$[p_8]$	$[p_0]$	$[p_8]$	$[p_9]$	$[p_1]$	$[p_{11}]$	$[p_3]$	$[p_2]$	$[p_{10}]$	$[p_{15}]$	$[p_7]$	$[p_6]$	$[p_{14}]$	$[p_4]$	$[p_{12}]$	$[p_{13}]$	$[p_5]$
$[p_9]$	$[p_0]$	$[p_9]$	$[p_{11}]$	$[p_2]$	$[p_{15}]$	$[p_6]$	$[p_4]$	$[p_{13}]$	$[p_7]$	$[p_{14}]$	$[p_{12}]$	$[p_5]$	$[p_8]$	$[p_1]$	$[p_3]$	$[p_{10}]$
$[p_{10}]$	$[p_0]$	$[p_{10}]$	$[p_{13}]$	$[p_7]$	$[p_3]$	$[p_9]$	$[p_{14}]$	$[p_4]$	$[p_6]$	$[p_{12}]$	$[p_{11}]$	$[p_1]$	$[p_5]$	$[p_{15}]$	$[p_8]$	$[p_2]$
$[p_{11}]$	$[p_0]$	$[p_{11}]$	$[p_{15}]$	$[p_4]$	$[p_7]$	$[p_{12}]$	$[p_8]$	$[p_3]$	$[p_{14}]$	$[p_5]$	$[p_1]$	$[p_{10}]$	$[p_9]$	$[p_2]$	$[p_6]$	$[p_{13}]$
$[p_{12}]$	$[p_0]$	$[p_{12}]$	$[p_1]$	$[p_{13}]$	$[p_2]$	$[p_{14}]$	$[p_3]$	$[p_{15}]$	$[p_4]$	$[p_8]$	$[p_5]$	$[p_9]$	$[p_6]$	$[p_{10}]$	$[p_7]$	$[p_{11}]$
$[p_{13}]$	$[p_0]$	$[p_{13}]$	$[p_3]$	$[p_{14}]$	$[p_6]$	$[p_{11}]$	$[p_5]$	$[p_8]$	$[p_4]$	$[p_{15}]$	$[p_1]$	$[p_{15}]$	$[p_3]$	$[p_7]$	$[p_5]$	$[p_2]$
$[p_{14}]$	$[p_0]$	$[p_{14}]$	$[p_5]$	$[p_{11}]$	$[p_{10}]$	$[p_4]$	$[p_{15}]$	$[p_1]$	$[p_{12}]$	$[p_3]$	$[p_8]$	$[p_6]$	$[p_1]$	$[p_5]$	$[p_2]$	$[p_{12}]$
$[p_{15}]$	$[p_0]$	$[p_{15}]$	$[p_7]$	$[p_8]$	$[p_{14}]$	$[p_1]$	$[p_9]$	$[p_6]$	$[p_5]$	$[p_{10}]$	$[p_2]$	$[p_{13}]$	$[p_{11}]$	$[p_2]$	$[p_{12}]$	$[p_3]$

Tabelle 22.11: Multiplikationstabelle der Polynome 3. Grades über $\mathbb{F}_2$ modulo dem irreduziblen Polynom $\pi_2 = x^4 + x^3 + 1$.

Der wichtige Schritt besteht also darin, ein irreduzibles Polynom über einem gegebenen Körper zu finden. Um alle normierten irreduziblen Polynome mit festen Grad über $\mathtt{GF}(p^m)$ zu bestimmen, kann man zunächst sämtliche normierten reduziblen Polynome mit festem Grad bestimmen und anschließend werden aus dieser Menge die zerlegbaren eliminiert. Wenn p groß ist, ist dies ein nicht praktiabler Weg[12].

Beispiel:

In diesem Beispiel erarbeiten wir alle irreduziblen Polynome vom Grad 4 über dem Feld $\mathtt{GF}(2^4)$.

Zunächst: Es gibt 16 verschiedene Polynome über $\mathtt{GF}(2^4)$ vom Grad 4:

$$\begin{aligned}
p_1(x) &= x^4 \\
p_2(x) &= x^4 + 1 \\
p_3(x) &= x^4 + x \\
p_4(x) &= x^4 + x + 1 \\
p_5(x) &= x^4 + x^2 \\
p_6(x) &= x^4 + x^2 + 1 \\
p_7(x) &= x^4 + x^2 + x \\
p_8(x) &= x^4 + x^2 + x + 1 \\
p_9(x) &= x^4 + x^3 \\
p_{10}(x) &= x^4 + x^3 + 1 \\
p_{11}(x) &= x^4 + x^3 + x \\
p_{12}(x) &= x^4 + x^3 + x + 1 \\
p_{13}(x) &= x^4 + x^3 + x^2 \\
p_{14}(x) &= x^4 + x^3 + x^2 + 1 \\
p_{15}(x) &= x^4 + x^3 + x^2 + x \\
p_{16}(x) &= x^4 + x^3 + x^2 + x + 1
\end{aligned}$$

Nun ist evident, dass diese Polynome über $\mathtt{GF}(2^4)$ reduzibel sind, wenn das Polynom einen Teiler vom Grad 1 oder 2 hat. Das heißt, wir berechnen alls Produkte der Form

$$\begin{aligned}
P_1 &= (x^3 + a_2x^2 + a_1x + a_0)(x + b_0) \\
P_2 &= (x^2 + a_1x + a_0)(x^2 + b_1x + b_0)
\end{aligned}$$

mit Koeffizienten $a_i, b_i \in GF(2)$.

Das erste Produkt ergibt:

$$P_1 = x^4 + (a_2 + b_0)x^3 + (a_1 + a_2b_0)x^2 + (a_0 + a_1b_0)x + a_0b_0$$

Wir müssen wieder alle 16 mögliche Fälle ausarbeiten:

[12] Siehe [144], chapter 3.

1. $a_2 = 0; a_1 = 0; a_0 = 0; b_0 = 0$
   $\Longrightarrow \quad P_1 = x^4 \quad \Longrightarrow P_1 = p_1(x)$
2. $a_2 = 0; a_1 = 0; a_0 = 0; b_0 = 1$
   $\Longrightarrow \quad P_1 = x^4 + x^3 \quad \Longrightarrow P_1 = p_9(x)$
3. $a_2 = 0; a_1 = 0; a_0 = 1; b_0 = 0$
   $\Longrightarrow \quad P_1 = x^4 + x \quad \Longrightarrow P_1 = p_3(x)$
4. $a_2 = 0; a_1 = 0; a_0 = 1; b_0 = 1$
   $\Longrightarrow \quad P_1 = x^4 + x^3 + x + 1 \quad \Longrightarrow P_1 = p_{12}(x)$
5. $a_2 = 0; a_1 = 1; a_0 = 0; b_0 = 0$
   $\Longrightarrow \quad P_1 = x^4 + x^2 \quad \Longrightarrow P_1 = p_5(x)$
6. $a_2 = 0; a_1 = 1; a_0 = 0; b_0 = 1$
   $\Longrightarrow \quad P_1 = x^4 + x^2 + x \quad \Longrightarrow P_1 = p_7(x)$
7. $a_2 = 0; a_1 = 1; a_0 = 1; b_0 = 0$
   $\Longrightarrow \quad P_1 = x^4 + x^2 + x \quad \Longrightarrow P_1 = p_7(x)$
8. $a_2 = 0; a_1 = 1; a_0 = 1; b_0 = 1$
   $\Longrightarrow \quad P_1 = x^4 + x^3 + x^2 + 1 \quad \Longrightarrow P_1 = p_{14}(x)$
9. $a_2 = 1; a_1 = 0; a_0 = 0; b_0 = 0$
   $\Longrightarrow \quad P_1 = x^4 + x^3 \quad \Longrightarrow P_1 = p_9(x)$
10. $a_2 = 1; a_1 = 0; a_0 = 0; b_0 = 1$
    $\Longrightarrow \quad P_1 = x^4 + x^2 \quad \Longrightarrow P_1 = p_5(x)$
11. $a_2 = 1; a_1 = 0; a_0 = 1; b_0 = 0$
    $\Longrightarrow \quad P_1 = x^4 + x^3 + x \quad \Longrightarrow P_1 = p_{11}(x)$
12. $a_2 = 1; a_1 = 0; a_0 = 1; b_0 = 1$
    $\Longrightarrow \quad P_1 = x^4 + x^2 + x + 1 \quad \Longrightarrow P_1 = p_8(x)$
13. $a_2 = 1; a_1 = 1; a_0 = 0; b_0 = 0$
    $\Longrightarrow \quad P_1 = x^4 + x^3 + x^2 \quad \Longrightarrow P_1 = p_{13}(x)$
14. $a_2 = 1; a_1 = 1; a_0 = 0; b_0 = 1$
    $\Longrightarrow \quad P_1 = x^4 + x \quad \Longrightarrow P_1 = p_3(x)$
15. $a_2 = 1; a_1 = 1; a_0 = 1; b_0 = 0$
    $\Longrightarrow \quad P_1 = x^4 + x^3 + x^2 + x \quad \Longrightarrow P_1 = p_{15}(x)$
16. $a_2 = 1; a_1 = 1; a_0 = 1; b_0 = 1$
    $\Longrightarrow \quad P_1 = x^4 + 1 \quad \Longrightarrow P_1 = p_2(x)$

Das zweite Produkt ist:

$$P_2 = x^4 + (b_1 + a_1)x^3 + (a_0 + b_0 + a - 1b_1)x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0$$

Auch hier sind die 16 mögliche Kombinationen der a_i, b_i zu berechnen:

1. $a_1 = 0; a_0 = 0; b_1 = 0; b_0 = 0$
   $\Longrightarrow \quad P_2 = x^4 \quad \Longrightarrow P_2 = p_1(x)$

2. $a_1 = 0; a_0 = 0; b_1 = 0; b_0 = 1$
   $\Longrightarrow \quad P_2 = x^4 + x^2 \quad \Longrightarrow P_2 = p_5(x)$

3. $a_1 = 0; a_0 = 0; b_1 = 1; b_0 = 0$
   $\Longrightarrow \quad P_2 = x^4 + x^3 \quad \Longrightarrow P_2 = p_9(x)$

4. $a_1 = 0; a_0 = 0; b_1 = 1; b_0 = 1$
   $\Longrightarrow \quad P_2 = x^4 + x^3 + x^2 \quad \Longrightarrow P_2 = p_{13}(x)$

5. $a_1 = 0; a_0 = 1; b_1 = 0; b_0 = 0$
   $\Longrightarrow \quad P_2 = x^4 + x^2 \quad \Longrightarrow P_2 = p_5(x)$

6. $a_1 = 0; a_0 = 1; b_1 = 0; b_0 = 1$
   $\Longrightarrow \quad P_2 = x^4 + 1 \quad \Longrightarrow P_2 = p_2(x)$

7. $a_1 = 0; a_0 = 1; b_1 = 1; b_0 = 0$
   $\Longrightarrow \quad P_2 = x^4 + x^3 + x^2 + x \quad \Longrightarrow P_2 = p_{15}(x)$

8. $a_1 = 0; a_0 = 1; b_1 = 1; b_0 = 1$
   $\Longrightarrow \quad P_2 = x^4 + x^3 + x + 1 \quad \Longrightarrow P_2 = p_{12}(x)$

9. $a_1 = 1; a_0 = 0; b_1 = 0; b_0 = 0$
   $\Longrightarrow \quad P_2 = x^4 + x^3 \quad \Longrightarrow P_2 = p_9(x)$

10. $a_1 = 1; a_0 = 0; b_1 = 0; b_0 = 1$
    $\Longrightarrow \quad P_2 = x^4 + x^3 + x^2 + x \quad \Longrightarrow P_2 = p_{15}(x)$

11. $a_1 = 1; a_0 = 0; b_1 = 1; b_0 = 0$
    $\Longrightarrow \quad P_2 = x^4 + x^2 \quad \Longrightarrow P_2 = p_5(x)$

12. $a_1 = 1; a_0 = 0; b_1 = 1; b_0 = 1$
    $\Longrightarrow \quad P_2 = x^4 + x \quad \Longrightarrow P_2 = p_3(x)$

13. $a_1 = 1; a_0 = 1; b_1 = 0; b_0 = 0$
    $\Longrightarrow \quad P_2 = x^4 + x^3 + x^2 \quad \Longrightarrow P_2 = p_{13}(x)$

14. $a_1 = 1; a_0 = 1; b_1 = 0; b_0 = 1$
    $\Longrightarrow \quad P_2 = x^4 + x^3 + x + 1 \quad \Longrightarrow P_2 = p_{12}(x)$

15. $a_1 = 1; a_0 = 1; b_1 = 1; b_0 = 0$
    $\Longrightarrow \quad P_2 = x^4 + x \quad \Longrightarrow P_2 = p_3(x)$

16. $a_1 = 1; a_0 = 1; b_1 = 1; b_0 = 1$
    $\Longrightarrow \quad P_2 = x^4 + x^2 + 1 \quad \Longrightarrow P_2 = p_6(x)$

Vergleicht man diese Resultate mit der Liste alle Polynome $p_1(x), \ldots, p_{16}(x)$, dann erkennt man, dass bis auf die drei Polynome

$$\begin{aligned} p_4(x) &= x^4 + x + 1 \\ p_{10}(x) &= x^4 + x^3 + 1 \\ p_{16}(x) &= x^4 + x^3 + x^2 + x + 1 \end{aligned}$$

jedes dieser 16 Polynome in Polynome niedrigeren Grades faktorisiert werden kann. Die drei Polynome, $p_4(x), p_{10}(x)$ $p_{16}(x)$ sind die drei irreduziblen Polynome über dem Feld $\mathtt{GF}(2^4)$.

Die oben definierte Multiplikation ist assoziativ, i.e.

$$p_1 \otimes (p_2 \otimes p_3) = (p_1 \otimes p_2) \otimes p_3$$

und es gibt ein neutrales Element – das konstante Polynom $\not\Vdash$ – so dass

$$p \otimes 1 = 1 \otimes p = p$$

Die sich nun stellende frage ist, wie man das multiplikative Inverse bezüglich dieser Multiplikationsoperation findet. Das bedeutet, für ein gegebenes Polynom $q(x)$ müssen die beiden Polynome finden, die

$$q(x)a(x) + \pi_1(x)c(x) = 1$$

erfüllen, also

$$q^{-1} = a(x) \bmod \pi_1(x).$$

Das Verfahren, das multiplikative Inverse zu finden, ist das gleiche wie für Zahlen, i.e. der erweiterte EUKLIDische Algorithmus für Polynome wird verwendet, das multiplikative inverse Polynome modulo einem gegebenen irreduziblen Polynom zu berechnen.

In dem endlichen Körper $\mathtt{GF}(2^8)$ – dieser spielt die grundlegende Rolle im **Rijndael Algorithmus** – betrachtet man Bytes der Form $b_i \in \mathbb{Z}_2,\ i = 0, \ldots 7$. Diese acht Bits können als Koeffizienten des folgenden Polynoms betrachtet werden:

$$\begin{aligned} b(x) &= b_7 \cdot x^7 + b_6 \cdot x^6 + b_5 \cdot x^5 + b_4 \cdot x_4 + b_3 \cdot x^3 + b_2 \cdot x_2 + b_1 \cdot x^1 + b_0 \\ &= \sum_{i=0}^{7} b_i \cdot x^i \end{aligned}$$

Beispielsweise kann auf diese Weise die Bitfolge `01 010 111`$\in \mathtt{GF}(2^8)$ durch das Polynom

$$x^6 + x^4 + x^2 + x + 1$$

repräsentiert werden. Üblicherweise verwendet man eine andere, kompaktere Notation. Interpretiert man den Bitstring `01 010 111` als eine Binärzahl, dann hat dieser den Wert

$$01\,010\,111_2 = 87_{10}$$

also 87 im Dezimalzahlensystem. Schreibt man diese Zahl in das Hexadezimalsystem, dann entspricht die Bitfolge `01 010 111` der Hexadezimalzahl `0x57`.

Die Multiplikation in $\mathtt{GF}(p^m)$ ist definiert als Polynommultiplikation **modulo einem gegebenen irreduziben Polynom** $\pi(x)$ **über** $\mathtt{GF}(p)$ **vom Grad** m. Für $p = 2$ und $m = 4$ gibt es genau drei solche Polynome:

$$\begin{aligned} \pi_1(x) &= x^4 + x + 1 \\ \pi_2(x) &= x^4 + x^3 + 1 \\ \pi_3(x) &= x^4 + x^3 + x^2 + x + 1 \end{aligned}$$

22.5 Linear Feedback Shift Register

Shift Register Sequenzen werden sowohl in der Kryptographie als auch in der Codierungstheorie sehr häufig eingesetzt. Strom Chiffren, die auf Shift Registern basieren, sind das Arbeitspferd der militärischen Kryptographie seit dem Aufkommen der Elektronik in der Kommunikation. Der Grund, warum uns im Rahmen der Kryptographie die Shiftregister interessieren, beruht darauf, dass Linear Feedback Shiftregister eine Hardware Implementierung der Multiplikation und Division von Polynomen über derm GALOIS-Feld $\mathtt{GF}(2^p)$ darstellen[13].

Im Prinzip ist ein **Linear Feedback Shift Register**[14] oder LFSR aus zwei Komponenten aufgebaut:

- einem **Schieberegister** (shift register)
- einer **Rückkopplungsfunktion** (feedback function)

Das Schieberegister ist eine Folge von Bits (siehe Abbildung [22.2]). In einer Hardware Realisierung besteht solch ein Register aus einer Reihe von Flip-Flops, die jeweils ein Bit speichern können. Die Länge eines Schieberegisters wird in Bit angegeben. Falls das Schieberegister n Bits speichern kann, nennt man es ein n-Bit Schieberegister. Jedesmal, wenn ein Bit benötigt wird, werden sämtliche Bits des Schieberegisters um eine Position nach rechts verschoben. Das neue Bit, das an der äußersten linken Position erzeugt wird, wird als Funktion der anderen Bits des Registers berechnet. Die Ausgabe des Registers ist ein Bit, meist das niedrigwertigste Bit. Die **Periode** des Schieberegisters ist die Länge der Ausgabesequenz bevor eine Wiederholung auftritt.

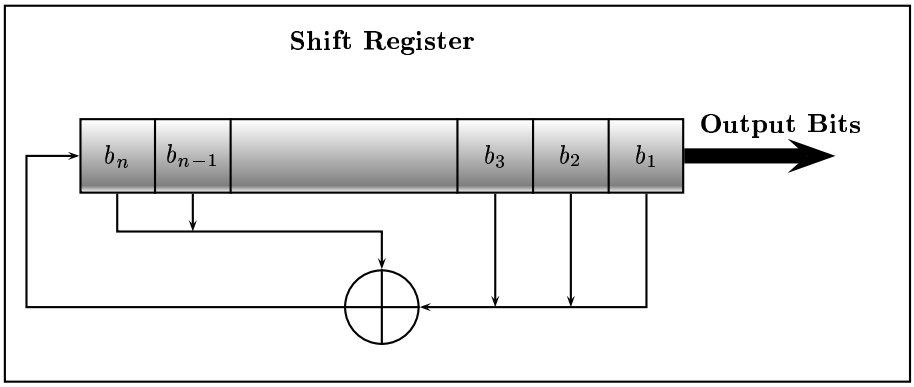


Abbildung 22.2: Aufbau eines Linear Feedback Shift Registers

Das Verhalten der Register wird durch einen Zähler gesteuert, in einer Hardware Implementierung wird dieser Zähler in der Regel durch die Uhr (Taktgeber) realisiert. Bei jedem Takt wird der Inhalt der Zellen des Registers um eine Position nach rechts verschoben und – für unsere momentanen Zwecke ausreichend

[13] Literatur zu Linear Feedback Shift Register findet man in [193, pp. 372], [215, Chap. 2.11] oder in [144, Chapter 6].

[14] Man nennt Feedback Shift Register auch *Rückkopplungsregister*.

– die XOR-Verknüpfung einer ausgewählten Teilmenge der Zellinhalte wird in die äußerste linke Zelle geschrieben. Üblicherweise wird pro Takt ein Ausgabebit generiert.

Etwas abstrakter formuliert, ein **Schieberegister** ist ein Gerät, e.g. ein Computerprogramm oder ein Chip, das die Elemente einer Folge

$$s = s_0, s_1, s_2, \dots$$

berechnet. Hierbei sind s_i Elemente einer Menge S. Die Berechnung geschieht derart, dass der Input $s_0, s_1, \dots, s_{n-1}$ vorgegeben ist[15] und der Rest der Folge wird gemäß einer Vorschrift

$$s_{j+n} = f\big(s_{j+n-1}, \dots s_j\big) \qquad \text{für alle } j \geq 0$$

berechnet. Dabei ist f eine Funktion, die man **erzeugende Funktion** oder **Feedback FunKtion** (Rückkopplungsfunktion) des Schieberegisters nennt.

Eine Rückkopplungsfunktion nennt man **linear**, wenn die Funktion Sequenzen erzeugt, deren Elemente aus einem Körper $\mathcal{K}$ sind und wenn die Funktion selbst linear ist. Außerdem nennt man die Rückkopplungsfunktion **homogen**, wenn sie keinen konstanten Term enthält.

Beispiel:

Linear Feedback Shift Register können (unter anderem) dazu verwendet werden, den Schlüsselstrom von Strom-Chiffren zu erzeugen. Dazu betrachten wir den Initial Vector bestehend aus den fünf Startwerten

$$s_0 = 0, s_1 = 1, s_2 = 0, s_3 = 0, s_4 = 0.$$

mit $s_i \in \mathbb{F}_2 = \mathbb{Z}_2 = \{0, 1\}$. Die Feedback Funktion ist:

$$s_{n+5} = s_{n+2} + s_n, \quad n = 0, 1, \dots$$

mit der Addition in $\mathbb{F}_2$.

[15] Dies nennt man mitunter auch *Initial Vector*.

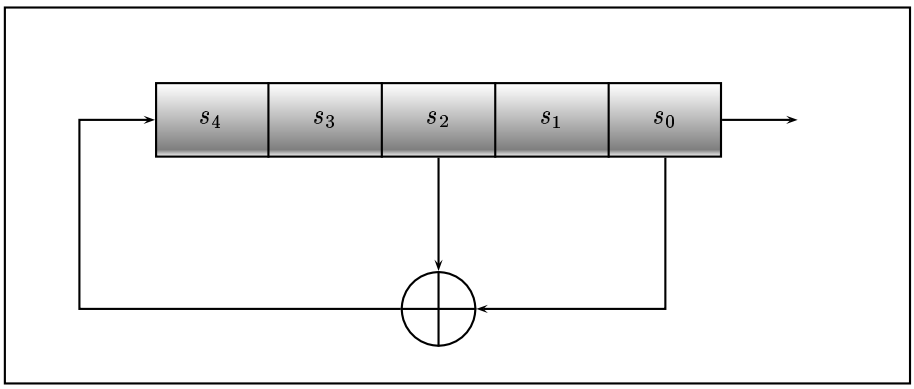


Abbildung 22.3: Linear Feedback Shift Register für die Rekursion $s_{n+5} \equiv (s_n + s_{n+2}) \bmod 2$.

Daraus leitet man die folgende Sequenz ab:

$$
\begin{array}{ll}
s_0 = 0 & s_{20} = s_{17} + s_{15} = 0 \\
s_1 = 1 & s_{21} = s_{18} + s_{16} = 0 \\
s_2 = 0 & s_{22} = s_{19} + s_{17} = 0 \\
s_3 = 0 & s_{23} = s_{20} + s_{18} = 1 \\
s_4 = 0 & s_{24} = s_{21} + s_{19} = 1 \\
s_5 = s_2 + s_0 = 0 & s_{25} = s_{22} + s_{20} = 0 \\
s_6 = s_3 + s_1 = 1 & s_{26} = s_{23} + s_{21} = 1 \\
s_7 = s_4 + s_2 = 0 & s_{27} = s_{24} + s_{22} = 1 \\
s_8 = s_5 + s_3 = 0 & s_{28} = s_{25} + s_{23} = 1 \\
s_9 = s_6 + s_4 = 1 & s_{29} = s_{26} + s_{24} = 0 \\
s_{10} = s_7 + s_5 = 0 & s_{30} = s_{27} + s_{25} = 1 \\
s_{11} = s_8 + s_6 = 1 & s_{31} = s_{28} + s_{26} = 0 \\
s_{12} = s_9 + s_7 = 1 & s_{32} = s_{29} + s_{27} = 1 \\
s_{13} = s_{10} + s_8 = 0 & s_{33} = s_{30} + s_{28} = 0 \\
s_{14} = s_{11} + s_9 = 0 & s_{34} = s_{31} + s_{29} = 0 \\
s_{15} = s_{12} + s_{10} = 1 & s_{35} = s_{32} + s_{30} = 0 \\
s_{16} = s_{13} + s_{11} = 1 & s_{36} = s_{33} + s_{31} = 0 \\
s_{17} = s_{14} + s_{12} = 1 & s_{37} = s_{34} + s_{32} = 1 \\
s_{18} = s_{15} + s_{13} = 1 & s_{38} = s_{35} + s_{33} = 0 \\
s_{19} = s_{16} + s_{14} = 1 & s_{39} = s_{36} + s_{34} = 0
\end{array}
$$

Diese Folge wiederholt sich nach 31 Termen.

Im allgemeinen betrachtet man eine lineare Rekursion der Länge k:

$$s_{n+k} = (a_n s_n + a_{n-1} s_{n-1} + \ldots + a_{n+k-1} s_{n+k-1} \tag{22.8}$$

wobei die Koeffizienten a_k aus $\mathbb{F}_2$ sind. Sind die Startwerte (Initial Vector (IV))

$$s_0, s_1, \ldots, s_{k-1}$$

festgelegt, dann können alle folgenden Werte der s_i aus der Rekursion (22.8) berechnet werden. Die resultierende Folge aus `0`en und `1`en können als Schlüssel für die Verschlüsselung benutzt werden.

Ein Vorteil dieser Methode liegt darin, dass ein Schlüssel mit großer Periode aus wenigen Informationen generiert werden kann. In dem obigen Beispiel haben wir den Initial Vector $(s_0, s_1, s_2, s_3, s_4) = (0, 1, 0, 0, 0)$ verwendet und die Koeffizienten sind $(a_0, a_1, a_2, a_3, a_4) = (1, 0, 1, 0, 0)$, was auf eine Periode von 31 führt. Das bedeutet also, 10 Bits werden benötigt, um 31 Bit zu erzeugen.

Man kann zeigen, dass die Rekursion

$$s_{n+31} = s_{n+3} + s_n$$

und ein beliebiger Initial Vector eine Folge erzeugt, die eine Periode $2^{31} - 1 = 2.147.483.647$ hat. In diesem Fall produzieren 62 Bits mehr als 2 Milliarden Bits für einen Schlüsselstrom. Dies ist ein großer Vorteil gegenüber dem One-Time Pad, bei dem alle 2 Milliarden Bits vorab übertragen werden müssen.

Um zu sehen, wie LFSR noch eingesetzt werden können, demonstrieren wir, wie Polynommultiplikation und –division mit linearen Shiftregistern realisiert werden können.

Polynommultiplikation

Wir betrachten zunächst die Multiplikation der beiden Polynome

$$\begin{aligned} f(x) &= x^5 + x^4 + x^2 + 1 \\ g(x) &= x^4 + x^2 + x + 1 \end{aligned}$$

in $GF(2)$. Wir erhalten:

$$\begin{aligned} f(x) \cdot g(x) &= (x^5 + x^4 + x^2 + 1) \cdot (x^4 + x^2 + x + 1) \\ &= x^9 + x^7 + x^6 + x^5 \\ &\quad + x^8 + x^6 + x^5 + x^4 \\ &\quad + x^6 + x^4 + x^3 + x^2 \\ &\quad + x^6 + x^4 + x^3 + x^2 \\ &\quad + x^4 + x^2 + x + 1 \\ &= x^9 + x^8 + x^7 + x^6 + x^4 + x^3 + x + 1 \\ &= h(x) \end{aligned}$$

Wir bilden diese Polynome auf `01` - Folgen ab und erhalten:

$$\begin{aligned} f(x) &\longmapsto \texttt{110101} \\ g(x) &\longmapsto \texttt{010111} \end{aligned}$$

Die Multiplikation dieser beiden Bitfolgen modulo 2 ergibt:

(1	1	0	1	0	1)	·	(1	0	1	1	1)
						1	1	0	1	0	1
					1	1	0	1	0	1	
				1	1	0	1	0	1		
⊕		1	1	0	1	0	1				
		1	1	1	1	0	1	1	0	1	1

Wird diese 01 Sequenz wieder in ein Polynom zurücktransformiert, erhalten wir:

$$1111011011 \longmapsto x^9 + x^8 + x^7 + x^6 + x^4 + x^3 + x + 1 = h(x)$$

Die Multiplikation dieser beiden Bitfolgen kann mit Hilfe von Linearen Feedback Shift Registern durchgeführt werden. Das obige Beispiel ist in der Abbildung [22.4] illustriert. Das Polynom $f(x) = x^5 + x^4 + x^2 + 1$ wird durch die fünf Register repräsentiert. Der Input Bitstrom ist das Polynom $g(x) \longmapsto 10111$.

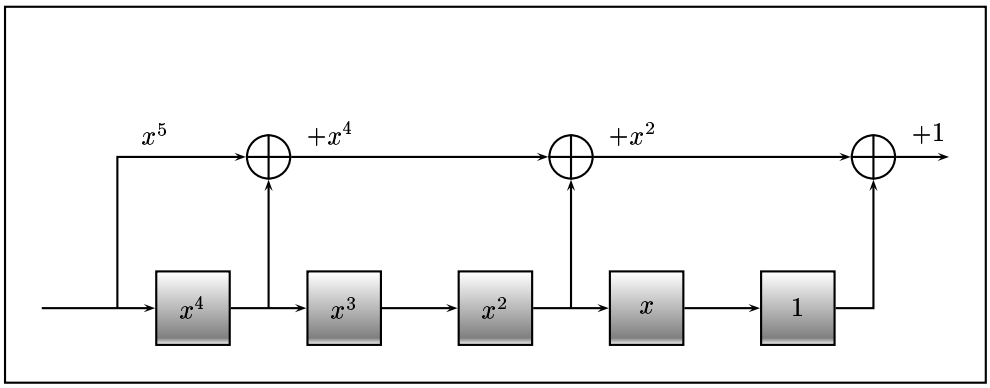


Abbildung 22.4: Multiplikation mit einem fünfstufigen Linear Feedback Shift Register

Time	Input	x^4	x^3	x^2	x	1	Output
Initial	11101	0	0	0	0	0	
T_1	01110	1	0	0	0	0	1
T_2	00111	0	1	0	0	0	1 1
T_3	00011	1	0	1	0	0	1 1 1
T_4	00001	1	1	0	1	0	1 1 1 1
T_5	00000	1	1	1	0	1	0 1 1 1 1
T_6	00000	0	1	1	1	0	1 0 1 1 1 1
T_7	00000	0	0	1	1	1	1 1 0 1 1 1 1
T_8	00000	0	0	0	1	1	0 1 1 0 1 1 1 1
T_9	00000	0	0	0	0	1	1 0 1 1 0 1 1 1 1
T_{10}	00000	0	0	0	0	0	1 1 0 1 1 0 1 1 1 1

Polynomdivision

Wir starten wieder mit zwei Polynomen

$$f(x) = x^5 + x^4 + x^3 + x + 1$$
$$g(x) = x^3 + x^2 + 1$$

mit Koeffizienten in $\mathbb{F}_2$. Dividiert man f durch g, dann erhält man:

$$f(x) : g(x) = x^2 + 1$$

mit Rest x^2. Betrachten wir wieder die Darstellung der Polynome als 01-Folgen, dann ergibt sich:

$$f(x) \longmapsto \texttt{111011}$$
$$g(x) \longmapsto \texttt{1101}$$

Die Division dieser beiden Bitfolgen wird folgendermaßen durchgeführt:

	1	1	1	0	1	1	:	1	1	0	1	= 1 0 1
$\oplus$	1	1	0	1								
	0	0	1	1								
			1	1	1							
			1	1	1	1						
$\oplus$			1	1	0	1						
			0	0	1	0						

Wir erhalten also: 111011 : 1101 = 101 mit Rest 010. Resubstituiert man diese Bitfolgen als Polynome sieht man, dass diese Operationen konsistent sind.

Um zu sehen, wie man mit Hilfe von Shiftregistern die Polynomdivision implementieren kann, betrachten wir die Abbildung [22.5]. Der Grad des Polynoms $g(x)$ ist drei. Daher hat das Shiftregister drei Speicherzellen, um jeweils 1 Bit zu speichern. Diese Speicherzellen repräsentieren die Koeffizienten des Polynoms $g(x) = x^3 + x^2 + 1$. Jeder nichtverschwindende Koeffizient in dem Teilerpolynom führt auf eine Rückkopplung mit einer XOR-Operation.

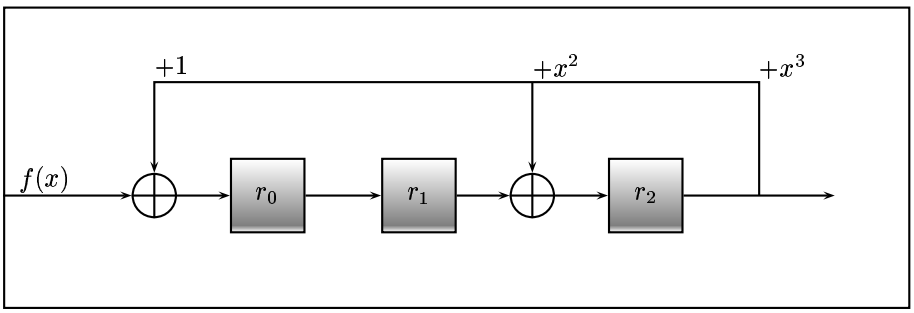


Abbildung 22.5: Division von Polynomen mit einem Linearen Feedback Shift Register.

Der Input ist das Polynom $f(x)$, repräsentiert durch die Bitsequenz 111011 in der Reihenfolge, dass das höchstwertige Bit zuerst in das LFSR einläuft. Der Startzustand des Registers ist 000.

Time	Input	r_0	r_1	r_2	Output
T_0	110111	0	0	0	
T_1	11011	1	0	0	0
T_2	1101	1	1	0	0
T_3	110	1	1	1	0
T_4	11	1	1	0	1
T_5	1	1	1	1	0
T_6		0	1	0	1

Aus dieser Verarbeitungsfolge erkennt man, dass nach der Verarbeitung der Inputsequenz der Output das Resultat der Division von f durch g repräsentiert und die drei Register r_1, r_2, r_3 enthalten den Rest der Division.

Eine andere Art und Weise, Linear Feedback Shift Register zu betrachten ist in Form von Gleichungen. Another way of considering linear feedback shift registers is in terms of equations. In general we have:

$$\begin{aligned} r_0[i] &= \text{in}[i] \oplus r_2[i-1] \\ r_1[i] &= r_0[i-1] \\ r_2[i] &= r_1[i-1] \oplus r_0[i-1] \\ \text{out}[i] &= r_2[i-1] \end{aligned}$$

for steps $i = 1, 2, \ldots 6$ and the initial condition

$$r_0[0] = r_1[0] = r_2[0] = 0$$

as well as the input stream

$$\begin{aligned} \text{in}[1] &= 1 \\ \text{in}[2] &= 1 \\ \text{in}[3] &= 1 \\ \text{in}[4] &= 0 \\ \text{in}[5] &= 1 \\ \text{in}[6] &= 1 \end{aligned}$$

Then we get:

$$\begin{aligned} r_0[1] &= \text{in}[1] \oplus r_2[0] = 1 \oplus 0 = 1 \\ r_1[1] &= r_0[0] = 0 \\ r_2[1] &= r_1[0] \oplus r_0[0] = 0 \oplus 0 = 0 \\ \text{out}[1] &= r_2[0] = 0 \end{aligned}$$

$$\begin{aligned} r_0[2] &= \text{in}[2] \oplus r_2[1] = 1 \oplus 0 = 1 \\ r_1[2] &= r_0[1] = 1 \\ r_2[2] &= r_1[1] \oplus r_0[1] = 0 \oplus 0 = 0 \\ \text{out}[2] &= r_2[1] = 0 \end{aligned}$$

$$
\begin{aligned}
r_0[3] &= \text{in}[3] \oplus r_2[2] = 1 \oplus 0 = 1 \\
r_1[3] &= r_0[2] = 1 \\
r_2[3] &= r_1[2] \oplus r_0[2] = 1 \oplus 0 = 1 \\
\text{out}[3] &= r_2[2] = 0
\end{aligned}
$$

$$
\begin{aligned}
r_0[4] &= \text{in}[4] \oplus r_2[3] = 0 \oplus 1 = 1 \\
r_1[4] &= r_0[3] = 1 \\
r_2[4] &= r_1[3] \oplus r_0[3] = 1 \oplus 1 = 0 \\
\text{out}[4] &= r_2[3] = 1
\end{aligned}
$$

$$
\begin{aligned}
r_0[5] &= \text{in}[5] \oplus r_2[4] = 1 \oplus 0 = 1 \\
r_1[5] &= r_0[4] = 1 \\
r_2[5] &= r_1[4] \oplus r_0[4] = 1 \oplus 0 = 1 \\
\text{out}[5] &= r_2[4] = 0
\end{aligned}
$$

$$
\begin{aligned}
r_0[6] &= \text{in}[6] \oplus r_2[5] = 1 \oplus 1 = 0 \\
r_1[6] &= r_0[5] = 1 \\
r_2[6] &= r_1[5] \oplus r_0[5] = 1 \oplus 1 = 0 \\
\text{out}[6] &= r_2[5] = 1
\end{aligned}
$$

Hence, we find after six iterations:

$$
\begin{aligned}
\text{out} &= [000101] \\
r &= [010]
\end{aligned}
$$

Kapitel 23

Linear rückgekoppelte Sequenzen

Folgen in endlichen Körpern, deren Terme in einfache Weise von den Vorgängern abhängen, spielen in vielen Anwendungen eine sehr wichtige Rolle. Solche Folgen sind einfach durch rekursive Verfahren zu erzeugen – dies ist sicherlich ein großer Vorteil hinsichtlich der Berechenbarkeit solcher Folgen – sie haben weiterhin auch sehr nützliche Strukturen. Von speziellem Interesse sind solche Sequenzen, bei denen die Terme *linear* von einer festen Anzahl der Vorgängerterme abhängen. Dies führt auf die sogenannten **linearen rückgekoppelten Sequenzen**. Solche Sequenzen werden beispielsweise in der

- Codierungstheorie
- Kryptographie
- Nachrichtentechnik

eingesetzt. In diesen Anwendungen ist der zugrunde liegende Körper $\mathbb{F}_2$, die Theorie kann jedoch ganz allgemein für jeden endlichen Körper entwickelt werden.

23.1 Feedback Shift Register, Periodizität

Die folgende Definition erklärt den Begriff einer linear rückgekoppelten Sequenz über einem beliebigen endlichen Körper $\mathbb{F}_q$.

Definition:

Sei k eine positive ganze Zahl und seien $a, a_0, a_1, \ldots, a_k$ gegebene Elemente eines endlichen Körpers $\mathbb{F}_q$. Eine Folge $s_0, s_1, \ldots$ von Elementen aus $\mathbb{F}_q$, die die Beziehung

$$s_{n+k} = a_{k-1}s_{n+k-1} + a_{k-2}s_{n+k-2} + \ldots + a_0 s_n + a, \ n = 0, 1, \ldots \tag{23.1}$$

erfüllt, heißt **linear rückgekoppelte Sequenz der Ordnung** k in $\mathbb{F}_q$.

Die k Folgenterme $s_0, s_1, \ldots, s_{k-1}$, welche die restlichen Folgenterme eindeutig bestimmen, nennt man **Startwerte** oder **Initial Value**. Eine Beziehung der Form (23.1) nennt man **lineare Rückkopplungsrelation** (der Ordnung k). Man spricht von einer **homogenen** linearen Rückkopplungsrelation, falls der konstante Term a Null ist, i.e. $a = 0$, andernfalls ist die Rückkopplungsrelation **inhomogen**. Die Folge

$$s_0, s_1, s_2, \ldots$$

selbst ist die homogene oder inhomogene lineare Rückkopplungsfolge in $\mathbb{F}_q$.

Die Erzeugung linearer Rückkopplungsfolgen kann mit Hilfe der **Feedback Shift Register** (FSR) durchgeführt werden. Wie in Kapitel [22.5] diskutiert, sind FSR elektronische Schaltkreise, die Informationen in Form von Elementen eines endlichen Körpers $\mathbb{F}_q$ verarbeiten können, wenn diese Elemente geeignet dargestellt werden. Betrachtet man Schieberegister über allgmeinen endlichen Körpern, dann werden vier Arten von Schaltelementen genutzt. Die Symbole, die die Elemente repräsentieren, sind in der Abbildung [23.1] dargestellt.

- Ein Addierer mit zwei Eingängen und einem Ausgang. Der Ausgang dieses Schaltelementes ist die Summe in $\mathbb{F}_q$ der beiden Eingänge.
- Ein konstanter Multiplikator mit einem Eingang. Der Ausgang dieses Schaltelements ist das Produkt in $\mathbb{F}_q$ eines konstanten Elements $a \in \mathbb{F}_q$ mit dem Input.
- Ein konstanter Addierer mit einem Eingang. Der Ausgang dieses Schaltelements ist die Summe in $\mathbb{F}_q$ eines konstanten Elements $a \in \mathbb{F}_q$ mit dem Input.
- Das vierte Schaltelement ist ein Verzögerungselement (delay element), in der Regel ein Flip-Flop. Dieses Element hat einen Eingang und einen Ausgang und wird durch einen externen Taktgeber (Clock) gesteuert. Dadurch wird aus dem Eingang zu einem Zeitpunkt der Ausgang einen Takt später.

Ein allgemeines Feedback Shift Register entsteht durch die Verbindung ei-

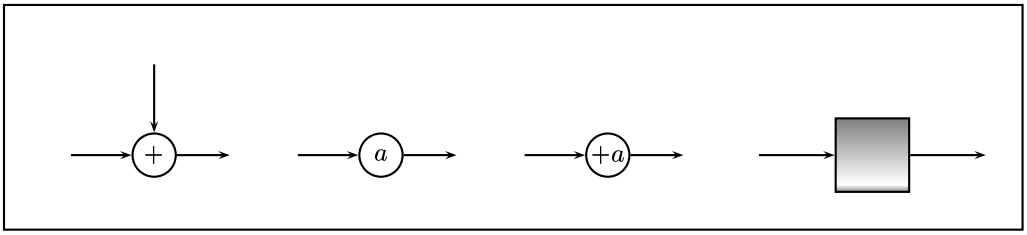


Abbildung 23.1: Die vier Bausteine von Feedback Shift Registern

ner endlichen Anzahl von Addierern, Multiplikatoren, konstanter Addierer und Delay-Elementen zu einem geschlossenen Zyklus in solch einer Weise, dass zwei Ausgänge niemals verbunden werden. Um lineare Rückkopplungsregister zu generieren, genügt es, die Komponenten in einer speziellen Art und Weise zu

verbinden. Ein Feedback Shift Register, das eine lineare Rückkopplungssequenz generiert, die die Gleichung (23.1) erfüllt, ist in der Abbildung [23.2] dargestellt.

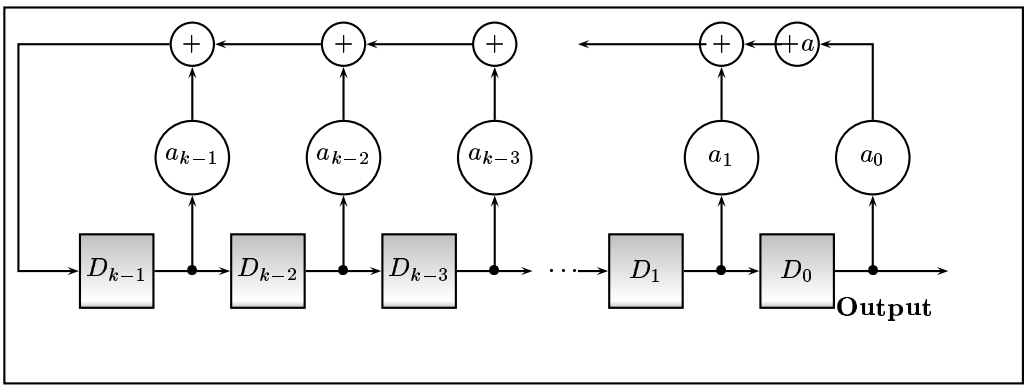


Abbildung 23.2: Die allgemeine Form eine Feedback Shift Registers

Zu Beginn enthält jedes Delay-Element $D_j, j = 0, 1, 2, \ldots, k-1$ den Startwert s_j. Denken wir uns die arithmetischen Operationen und den Transfer entlang der Verdrahtung verzögerungsfrei ausgeführt, dann enthält nach einer Zeiteinheit jedes Delay-Element den Wert s_{j+1}. Fährt man auf diese Weise fort, dann erkennt man, dass der Output des Shiftregisters die Folge $s_0, s_1, s_2, \ldots$ ist. Die Ausgabe erfolgt mit einem Zeichen pro Zeitintervall. In den meisten Anwendungen ist die Rückkopplungssequenz homogen, in diesem Fall ist der konstante Addierer nicht notwendig.

Beispiel [23.75] Betrachte die folgende lineare Rückkopplungssequenz in $\mathbb{F}_5$:

$$s_{n+6} = s_{n+5} + 2s_{n+4} + s_{n+1} + 3s_n, \quad n = 0, 1, 2, \ldots \tag{23.2}$$

Das in der Abbildung [23.3] dargestellte Feedback Register repräsentiert die

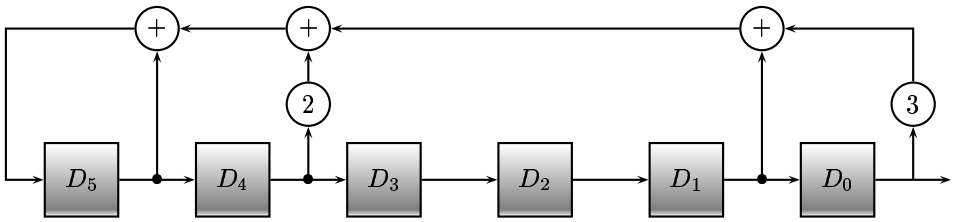


Abbildung 23.3: Feedback Shift Register der Sequenz $s_{n+6} = s_{n+5} + 2s_{n+4} + s_{n+1} + 3s_n$

lineare Rückkopplungsrelation (23.2).

Beispiel [23.76] In einem weiteren Beispiel betrachten wir die homogene lineare Rückkopplungsrelation

$$s_{n+7} = s_{n+4} + s_{n+3} + s_{n+2} + s_n, \quad n = 0, 1, 2, \ldots \tag{23.3}$$

in dem Körper $\mathbb{F}_2$.

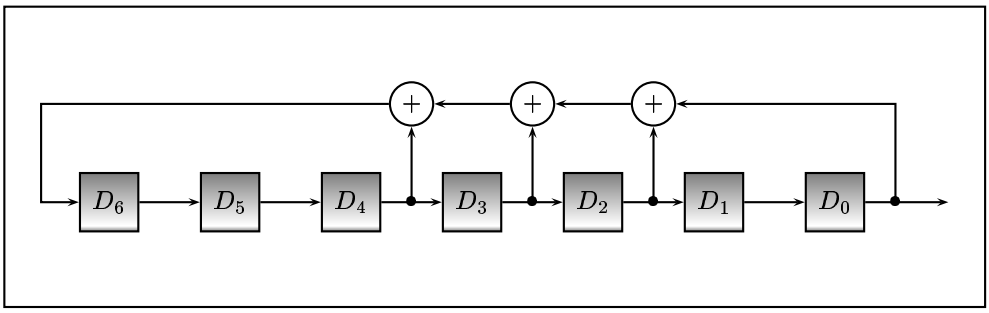


Abbildung 23.4: Linear Feedback Shift Register für die Sequenz (23.3)

Ein Lineares Feedback Shift Register, das mit der Rückkopplungssequenz (23.3) korrespondiert, ist in der Abbildung [23.4] dargestellt. Da die Multiplikation mit einer Konstanten in $\mathbb{F}_2$ die Elemente in $\mathbb{F}_2$ entweder annihiliert oder erhält, kann der Effekt eines konstanten Multiplikators in diesem Fall durch eine Verdrahtung oder der Weglassung der Verdrahtung simuliert werden. Daher erfordert ein lineares Feedback Shift Register zur Erzeugung binärer linearer Rückkopplungssequenzen lediglich die Elemente Delay-Element, Addierer und Verdrahtungen.

Sei $s_0, s_1, \ldots$ eine lineare Rückkopplungssequenz der Ordnung k in $\mathbb{F}_q$, die die Relation (23.1) erfüllt. Wie erwähnt, kann solch eine Folge durch das Feedback Shift Register der Abbildung [23.2] generiert werden. Wenn n eine nichtnegative, ganze Zahl ist, dann enthält das Delay-Element $D_j, j = 0, 1, \ldots, k-1$ nach n Zeiteinheiten das Element s_{n+j}. Es ist daher naheliegend, den k-elementigen Zeilenvektor

$$\vec{s_n} = (s_n, s_{n+1}, \ldots, s_{n+k-1})$$

den **n-ten Zustandsvektor** der linearen Rückkopplungssequenz (oder des Feedback Shift Registers) zu nennen. Den Zustandsvektor

$$\vec{s_0} = (s_0, s_1, \ldots, s_{k-1})$$

nennt man Initial Vektor.

Beispiel [**23.77**] Betrachte die homogene linear rückgekoppelte Sequenz der Ordnung 3 über $\mathbb{F}_2$:

$$s_{n+3} = s_n + s_{n+1}.$$

Diese Rückkopplungsrelation wird durch das LFSR aus der Abbildung [23.5] repräsentiert.

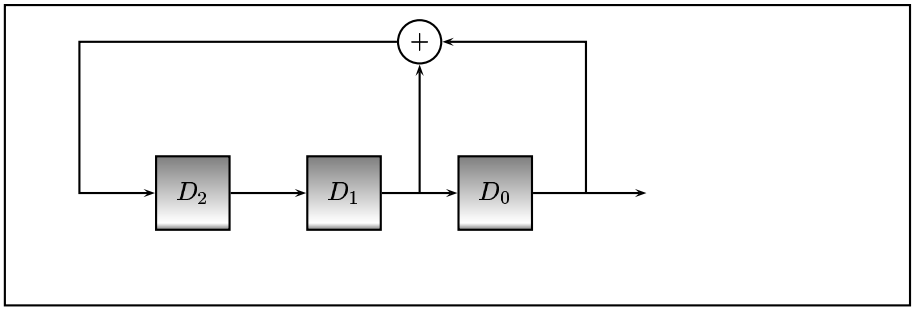


Abbildung 23.5: Linear Feedback Shift Register zur Erzeugung der Sequenz $s_{n+3} = s_n + s_{n+1}$ über $\mathbb{F}_2$.

Startet man mit dem Initial Vektor $\vec{s}_0 = (s_0, s_1, s_2) = (1, 0, 0)$, dann ergeben sich die folgenden Zustandsvektoren:

$$\begin{aligned}
\vec{s}_0 &= (s_0, s_1, s_2) = (1, 0, 0) \\
\vec{s}_1 &= (s_1, s_2, s_3) = (0, 0, 1) \\
\vec{s}_2 &= (s_2, s_3, s_4) = (0, 1, 0) \\
\vec{s}_3 &= (s_3, s_4, s_5) = (1, 0, 1) \\
\vec{s}_4 &= (s_4, s_5, s_6) = (0, 1, 1) \\
\vec{s}_5 &= (s_5, s_6, s_7) = (1, 1, 1) \\
\vec{s}_6 &= (s_6, s_7, s_8) = (1, 1, 0) \\
\vec{s}_7 &= (s_7, s_8, s_9) = (1, 0, 0) = \vec{s}_0.
\end{aligned}$$

In der anderen Schreibweise ergibt sich die Sequenz:

$$1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ \ldots \tag{23.4}$$

Jeder andere Initial Vektor – mit Ausnahme des Nullvektors $(0, 0, 0)$ – führt auf exakt die gleiche Sequenz, da jeder andere Initial Vektor lediglich eine Verschiebung der Sequenz zur Folge hat.

Aus der Folge der Zustandsvektoren oder aus der Sequenz (23.4) erkennt man, dass – bis auf den Nullvektor – alle möglichen Zustände angenommen werden, bevor eine Wiederholung auftritt. Das heißt, die Zustandsvektoren $\vec{s}_i$ durchlaufen alle möglichen Zusände. Wir haben in diesem Beispiel den folgenden Sachverhalt: Die rückgekoppelte Sequenz über $\mathbb{F}_2$ ist von der Ordnung 3, dies führt auf $2^3 - 1 = 7$ verschiedene Zustände der Sequenz, bis eine Wiederholung auftritt.

Betrachte die homogene linear rückgekoppelte Sequenz der Ordnung 4 über $\mathbb{F}_2$:

$$s_{n+4} = s_n + s_{n+1}.$$

Diese Sequenz führt auf die folgenden Zustandsvektoren:

$$\begin{aligned}
\vec{s}_0 &= (s_0, s_1, s_2, s_3) = (0,0,0,1) & \vec{s}_8 &= (s_8, s_9, s_{10}, s_{11}) = (0,1,0,1)\\
\vec{s}_1 &= (s_1, s_2, s_3, s_4) = (0,0,1,0) & \vec{s}_9 &= (s_9, s_{10}, s_{11}, s_{12}) = (1,0,1,1)\\
\vec{s}_2 &= (s_2, s_3, s_4, s_5) = (0,1,0,0) & \vec{s}_{10} &= (s_{10}, s_{11}, s_{12}, s_{13}) = (0,1,1,1)\\
\vec{s}_3 &= (s_3, s_4, s_5, s_6) = (1,0,0,1) & \vec{s}_{11} &= (s_{11}, s_{12}, s_{13}, s_{14}) = (1,1,1,1)\\
\vec{s}_4 &= (s_4, s_5, s_6, s_7) = (0,0,1,1) & \vec{s}_{12} &= (s_{12}, s_{13}, s_{14}, s_{15}) = (1,1,1,0)\\
\vec{s}_5 &= (s_5, s_6, s_7, s_8) = (0,1,1,0) & \vec{s}_{13} &= (s_{13}, s_{14}, s_{15}, s_{16}) = (1,1,0,0)\\
\vec{s}_6 &= (s_6, s_7, s_8, s_9) = (1,1,0,1) & \vec{s}_{14} &= (s_{14}, s_{15}, s_{16}, s_{17}) = (1,0,0,0)\\
\vec{s}_7 &= (s_7, s_8, s_9, s_{10}) = (1,0,1,0) & \vec{s}_{15} &= (s_{15}, s_{16}, s_{17}, s_{18}) = (0,0,0,1)
\end{aligned}$$

Diese Sequenz generiert also folgende Bitfolge:

$$\underbrace{0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1}_{15}\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ \ldots$$

Auch diese Sequenz hat die Eigenschaft, dass $2^4 - 1 = 15$ verschiedene Zustände auftreten, bevor eine Wiederholung auftritt.

Wie dieses letzte beispiel zeigt, ist es eine charakteristische Eigenschaft von linearen Rückkopplungssequenzen in endlichen Körpern, dass – nach einem möglichen irregulären Verhalten am Anfang – die Folge irgendwann periodisch wird. Bevor wir diese Eigenschaften im Detail untersuchen, führen wir zunächst die adäquate Terminologie ein.

Definition:

Sei S eine beliebige, nichtleere Menge und sei $s_0, s_1, \ldots$ eine Folge von Elementen aus S. Falls es ganze Zahlen $r > 0$ und $n_0 \geq 0$ gibt, so dass $s_{n+r} = s_n$ für alle $n \geq n_0$, dann heißt die Folge **ultimativ periodisch** und r heißt **Periode** der Folge. Die kleinste Zahl unter allen möglichen Perioden einer ultimativ periodischen Folge heißt **kleinste Periode** der Folge.

Beispiel [**23.78**] Betrachte die Folge in $\mathbb{F}_2$:

$$0\ 0\ 0\underbrace{0\ 0\ 1\ 1}_{4}0\ 0\ 1\ 1\ \underbrace{\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1}_{8}\ \ldots$$

Diese Folge hat die Perioden 4, 8, 12, 16, usw. Die kleinste Periode ist 0011 also 4. Weiterhin erkennt man, dass sich die Folge erst ab dem dritten Glied wiederholt, sie hat also einen irregulären Anfang. Oder in obiger Terminologie, für alle $n \geq 3$ ist $s_{n+4} = s_n$; die Folge ist also ultimativ periodisch.

Lemma [1]:

Jede Periode einer ultimativ periodischen Folge ist teilbar durch die kleinste Periode.

Beweis:

Sei r eine beliebige Periode der ultimativ periodischen Sequenz $s_0, s_1, \ldots$, und sei r_1 die kleinste Periode. Dann haben wir also

$$s_{n+r} = s_n \quad \forall\, n \geq n_0$$

und

$$s_{n+r_1} = s_n \quad \forall\, n \geq n_1$$

mit geeigneten nichtnegativen ganzen Zahlen n_0, n_1. Ist r nicht teilbar durch r_1, dann können wir setzen:

$$r = mr_1 + t$$

mit ganzen Zahlen $m \geq 1$ und $0 < t < r_1$. Dann erhalten wir für alle $n \geq \max(n_0, n_1)$:

$$\begin{aligned} s_n &= s_{n+r} \\ &= s_{n+mr_1+t} \\ &= s_{n+(m-1)r_1+t} \\ &= \quad \vdots \\ &= s_{n+t}. \end{aligned}$$

Daher ist t eine Periode der Folge mit $0 < t < r_1$, was der Annahme widerspricht, dass r_1 die kleinste Periode der Folge ist.

Definition [23.25]:

Eine ultimativ periodische Folge $s_0, s_1, \ldots$ mit kleinster Periode r heißt **periodisch**, wenn $s_{n+r} = s_n$ für alle $n = 0, 1, \ldots$.

Wenn $s_0, s_1, \ldots$ eine ultimativ periodische Folge mit kleinster Periode r ist, dann nennt man die kleinste nichtnegative ganze Zahl n_0 mit $s_{n+r} = s_n$ für alle $n \geq n_0$ die **Vorperiode**. Eine Folge ist genau dann periodisch, wenn die Vorperiode 0 ist.

Wir kehren nun zu den linearen Rückkopplungssequenzen in endlichen Körpern zurück und untersuchen die grundlegenden Resultate der Periodizität solcher Folgen.

Theorem [17]:

Sei $\mathbb{F}_q$ ein endlicher Körper und k eine positive ganze Zahl. Dann ist jede lineare Rückkopplungssequenz der Ordnung k in $\mathbb{F}_q$ ultimativ periodisch mit kleinster Periode r, die die Bedingung $r \leq q^k$ erfüllt wenn die Folge inhomogen ist und $r \leq q^k - 1$ falls die Folge homogen ist.

Beweis:

Zunächst ist anzumerken, dass es in $\mathbb{F}_q$ genau q^k verschiedene k-Tupel von Elementen aus $\mathbb{F}_q$ gibt. Betrachten wir daher die Zustandsvektoren

$$\vec{s}_m \text{ mit } 0 \leq m \leq q^k$$

einer gegebenen linearen Rückkopplungssequenz der Ordnung k in $\mathbb{F}_q$, dann folgt

$$\vec{s}_j = \vec{s}_i$$

für zwei i, j mit $0 \leq i < j \leq q^k$.

Beispiel:

Um zu illustrieren, was dies bedeutet, betrachten wir den Fall $\mathbb{F}_2$ mit $k = 4$. In diesem Fall gibt es $q^k = 2^4 = 16$ verschiedene 4-Tupel:

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

Die Zustandsvektoren $\vec{s}_m$ mit $0 \leq m \leq q^k = 2^4 = 16$ sind die folgenden 4-Tupel:

$$\begin{aligned}
\vec{s}_0 &= (s_0, s_1, s_2, s_3) \\
\vec{s}_1 &= (s_1, s_2, s_3, s_4) \\
\vec{s}_2 &= (s_2, s_3, s_4, s_5) \\
\vec{s}_3 &= (s_3, s_4, s_5, s_6) \\
\vec{s}_4 &= (s_4, s_5, s_6, s_7) \\
\vec{s}_5 &= (s_5, s_6, s_7, s_8) \\
\vec{s}_6 &= (s_6, s_7, s_8, s_9) \\
\vec{s}_7 &= (s_7, s_8, s_9, s_{10}) \\
\vec{s}_8 &= (s_8, s_9, s_{10}, s_{11}) \\
\vec{s}_9 &= (s_9, s_{10}, s_{11}, s_{12}) \\
\vec{s}_{10} &= (s_{10}, s_{11}, s_{12}, s_{13}) \\
\vec{s}_{11} &= (s_{11}, s_{12}, s_{13}, s_{14}) \\
\vec{s}_{12} &= (s_{12}, s_{13}, s_{13}, s_{14}) \\
\vec{s}_{13} &= (s_{13}, s_{14}, s_{15}, s_{16}) \\
\vec{s}_{14} &= (s_{14}, s_{15}, s_{16}, s_{17}) \\
\vec{s}_{15} &= (s_{15}, s_{16}, s_{17}, s_{18}) \\
\vec{s}_{16} &= (s_{16}, s_{17}, s_{18}, s_{19})
\end{aligned}$$

Dies sind aber 17 verschiedene 4-Tupel, da $0 \leq m \leq 16$, das heißt, es muss zwei Zustandsvektoren s_i und s_j in dieser Menge geben, die gleich sind.

Wendet man die lineare Rückkopplungsrelation an und benutzt die vollständige Induktion, dann folgt,

$$\vec{s}_{n+j-i} = \vec{s}_n \quad \forall\, n \geq i$$

i.e. die lineare Rückkopplungsequenz selbst ist ultimativ periodisch mit kleinster Periode

$$r \leq j - i \leq q^k.$$

Im Fall, dass die lineare Rückkopplungssequenz homogen ist und keiner der Zustandsvektoren ist der Nullvektor, kann man die gleiche Argumentationskette nachvollziehen, wobei q^k durch $q^k - 1$ ersetzt werden muss, um $r \leq q^k - 1$ zu erhalten. Falls jedoch einer der Zustandsvektoren der Nullvektor ist, sind alle folgenden Zustandsvektoren Nullvektoren und die resultierende Sequenz hat die kleinste Periode $r = 1 \leq q^k - 1$.

Beispiel [23.79]

Die lineare Rückkopplungssequenz erster Ordnung $s_0, s_1, \ldots$ in $\mathbb{F}_p$, p Primzahl, mit

$$s_{n+1} = s_n + 1 \quad n = 0, 1, 2, \ldots$$

und beliebigem $s_0 \in \mathbb{F}_p$ zeigt, dass die obere Grenze für r in obigem Theorem angenommen wird. Fall $\mathbb{F}_q$ irgendein beliebiger endlicher Körper ist und g ist ein primitives Element dann hat die lineare Rückkopplungssequenz

$$s_0, s_1, \ldots \text{ in } \mathbb{F}_q$$

mit

$$s_{n+1} = g s_n, n = 0, 1, \ldots \quad s_0 \neq 0$$

die kleinste Periode $r = q - 1$, *i.e.* die obere Grenze kann auch im homogenen Fall angenommen werden.

Wir werden später sehen, dass in jedem endlichen Körper $\mathbb{F}_q$ und für jedes $k \geq 1$ lineare Rückkopplungssequenzen der k-ten Ordnung existieren, mit kleinster Periode $r = q^k - 1$.

Beispiel [23.80]

Für eine homogene lineare Rückkopplungssequenz erster Ordnung in $\mathbb{F}_q$ teilt die kleinste Periode $q - 1$. Falls $k \geq 2$ dann muß die kleinste Periode einer homogenen linearen Rückkopplungssequenz der Ordnung k nicht $q^k - 1$ teilen.

Betrachte zum Beispiel die Sequenz $s_0, s_1, \ldots \in \mathbb{F}_5$ mit

$$s_0 = 0, s_1 = 1$$

und

$$s_{n+2} = s_{n+1} + s_n.$$

In diesem Beispiel ist also $q = 5$ und die Rückkopplungsrelation ist homogen und von der Ordnung 2. Damit ist $q^k - 1 = 24$. Dann gilt:

$$
\begin{aligned}
s_2 &= s_1 + s_0 = 0 + 1 = 1 \\
s_3 &= s_2 + s_1 = 1 + 1 = 2 \\
s_4 &= s_3 + s_2 = 2 + 1 = 3 \\
s_5 &= s_4 + s_3 = 3 + 2 = 0 \\
s_6 &= s_5 + s_4 = 0 + 3 = 3 \\
s_7 &= s_6 + s_5 = 3 + 0 = 3 \\
s_8 &= s_7 + s_6 = 3 + 3 = 1 \\
s_9 &= s_8 + s_7 = 1 + 3 = 4 \\
s_{10} &= s_9 + s_8 = 4 + 1 = 0 \\
s_{11} &= s_{10} + s_9 = 0 + 4 = 4
\end{aligned}
$$

$$
\begin{aligned}
s_{12} &= s_{11} + s_{10} = 4 + 0 = 4 \\
s_{13} &= s_{12} + s_{11} = 4 + 4 = 3 \\
s_{14} &= s_{13} + s_{12} = 3 + 4 = 2 \\
s_{15} &= s_{14} + s_{13} = 2 + 3 = 0 \\
s_{16} &= s_{15} + s_{14} = 0 + 2 = 2 \\
s_{17} &= s_{16} + s_{15} = 2 + 0 = 2 \\
s_{18} &= s_{17} + s_{16} = 2 + 2 = 4 \\
s_{19} &= s_{18} + s_{17} = 4 + 2 = 1 \\
s_{20} &= s_{19} + s_{18} = 1 + 4 = 0 \\
s_{21} &= s_{20} + s_{19} = 0 + 1 = 1 \\
s_{22} &= s_{21} + s_{20} = 1 + 0 = 1 \\
s_{23} &= s_{22} + s_{21} = 1 + 1 = 2.
\end{aligned}
$$

Die kleinste Periode dieser Sequenz ist 20, was kein Teiler von $q^k - 1 = 24$ ist.

Ein wichtiges hinreichendes Kriterium für die Periodizität einer linearen Rückkopplungssequenz wird durch das folgende Resultat gegeben:

Theorem [18]:

Falls $s_0, s_1, \ldots$ eine lineare Rückkopplungssequenz in einem endlichen Körper ist, die die lineare Rückkopplungsrelation (23.1)

$$s_{n+k} = a_{k-1}s_{n+k-1} + a_{k-2}s_{n+k-2} + \ldots + a_0 s_n + a,\ n = 0, 1, \ldots$$

erfüllt, und der Koeffizient a_0 ist nicht 0, dann ist die Folge $s_0, s_1 \ldots$ periodisch.

Beweis:

Als Folge des Theorems [17] ist die gegebene Sequenz ultimativ periodisch. Sei r die kleinste Periode und n_0 die Vorperiode. Dann ist also

$$s_{n+r} = s_n \quad \forall n \geq n_0.$$

Angenommen wir haben $n_0 \geq 1$. Aus der Rekursionsrelation

$$s_{n+k} = a_{k-1}s_{n+k-1} + a_{k-2}s_{n+k-2} + \ldots + a_0 s_n + a, \quad n = 0, 1, \ldots$$

folgt durch Auflösen nach dem Term $a_0 s_n$:

$$a_0 s_n = s_{n+k} - a_{k-1}s_{n+k-1} - \ldots - a_1 s_{n-1} - a.$$

Da gemäß Annahme $a_0 \neq 0$ folgt:

$$s_n = a_0^{-1}\Big(s_{n+k} - a_{k-1}s_{n+k-1} - \ldots - a_1 s_{n+1} - a\Big).$$

Setzen wir $n = n_0 + r - 1$, dann liefert die Rekursionsrelation

$$s_{n_0+r-1} = a_0^{-1}\Big(s_{n_0+k-1+r} - a_{k-1}s_{n_0+k-2+r} - \ldots - a_1 s_{n_0+r} - a\Big).$$

Da r die kleinste Periode ist, gilt auch:

$$s_{n_0+r-1} = a_0^{-1}\Big(s_{n_0+k-1} - a_{k-1}s_{n_0+k-2} - \ldots - a_1 s_{n_0} - a\Big).$$

Führt man die exakt gleiche Rechnung mit $n = n_0 - 1$ durch, erhält man den gleichen Ausdruck für s_{n_0-1}. Damit haben wir die Gleichheit

$$s_{n_0-1+r} = s_{n_0-1}.$$

Dies ist aber ein Widerspruch zur Definition der Vorperiode.

Beispiel [23.81]

Betrachte die Rekursion

$$s_{n+4} = s_{n+2} + s_{n+1}$$

über $\mathbb{F}_2$. In dieser Rückkopplungsrelation ist der Term $a_0 = 0$. Diese lineare Rückkopplungsrelation führt mit dem Initial Vektor

$$s_0 = 0, s_1 = 0, s_2 = 0, s_3 = 1$$

auf die Folge:

$$\begin{array}{ll} s_0 = 0 & s_8 = 0 \\ s_1 = 0 & s_9 = 0 \\ s_2 = 0 & s_{10} = 1 \\ s_3 = 1 & s_{11} = 0 \\ s_4 = 0 & s_{12} = 1 \\ s_5 = 1 & s_{13} = 1 \\ s_6 = 1 & s_{14} = 1 \\ s_7 = 1 & s_{15} = 0. \end{array}$$

Dies führt daher auf die Zustandsvektoren:

$$\begin{aligned} \vec{s_0} &= (s_0, s_1, s_2, s_3) = (0, 0, 0, 1) \\ \vec{s_1} &= (s_1, s_2, s_3, s_4) = (0, 0, 1, 0) \\ \vec{s_2} &= (s_2, s_3, s_4, s_5) = (0, 1, 0, 1) \\ \vec{s_3} &= (s_3, s_4, s_5, s_6) = (1, 0, 1, 1) \\ \vec{s_4} &= (s_4, s_5, s_6, s_7) = (0, 1, 1, 1) \\ \vec{s_5} &= (s_5, s_6, s_7, s_8) = (1, 1, 1, 0) \\ \vec{s_6} &= (s_6, s_7, s_8, s_9) = (1, 1, 0, 0) \\ \vec{s_7} &= (s_7, s_8, s_9, s_{10}) = (1, 0, 0, 1) \\ \vec{s_8} &= (s_8, s_9, s_{10}, s_{11}) = (0, 0, 1, 0) = \vec{s_1} \\ \vec{s_9} &= (s_9, s_{10}, s_{11}, s_{12}) = (0, 1, 0, 1) = \vec{s_2} \end{aligned}$$

Diese Folge hat die Vorperiode 1 und kleinste Periode 7. Diese Folge ist also ultimativ periodisch aber nicht periodisch.

Die Rekursion

$$s_{n+4} = s_{n+2} + s_n$$

über $\mathbb{F}_2$ hat nun $a_0 \neq 0$. Sie führt mit dem Initial Vektor

$$s_0 = 0, s_1 = 0, s_2 = 0, s_3 = 1$$

auf die Folge:

$$\begin{array}{ll} s_0 = 0 & s_8 = 0 \\ s_1 = 0 & s_9 = 1 \\ s_2 = 0 & s_{10} = 0 \\ s_3 = 1 & s_{11} = 1 \\ s_4 = 0 & s_{12} = 0 \\ s_5 = 1 & s_{13} = 0 \\ s_6 = 0 & s_{14} = 0 \\ s_7 = 0 & s_{15} = 1 \end{array}$$

Diese Folge hat die Vorperiode 0, ist also periodisch. Die Zustandsvektoren sind für diese Sequenz:

$$\begin{aligned}
\vec{s_0} &= (s_0, s_1, s_2, s_3) = (0,0,0,1)\\
\vec{s_1} &= (s_1, s_2, s_3, s_4) = (0,0,1,0)\\
\vec{s_2} &= (s_2, s_3, s_4, s_5) = (0,1,0,1)\\
\vec{s_3} &= (s_3, s_4, s_5, s_6) = (1,0,1,0)\\
\vec{s_4} &= (s_4, s_5, s_6, s_7) = (0,1,0,0)\\
\vec{s_5} &= (s_5, s_6, s_7, s_8) = (1,0,0,0)\\
\vec{s_6} &= (s_6, s_7, s_8, s_9) = (0,0,0,1) = \vec{s}_0\\
\vec{s_7} &= (s_7, s_8, s_9, s_{10}) = (0,0,1,0) = \vec{s}_1
\end{aligned}$$

Sei $s_0, s_1, \ldots$ eine homogene lineare Rekursionssequenz der Ordnung k über einem endlichen Körper $\mathbb{F}_q$, die die lineare Rekursionsrelation

$$s_{n+k} = a_{k-1}s_{n+k-1} + a_{k-2}s_{n+k-2} + \ldots + a_0 s_n, \quad n = 0, 1, \ldots \tag{23.5}$$

erfüllt. Die Koeffizienten a_j sind Elemente von $\mathbb{F}_q$, $0 \le j \le k-1$. Wir assoziieren mit dieser linearen Rückkopplungssequenz die $k \times k$-Matrix $\mathbf{A}$ über $\mathbb{F}_q$, definiert durch:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & a_0 \\ 1 & 0 & 0 & \cdots & 0 & a_1 \\ 0 & 1 & 0 & \cdots & 0 & a_2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & a_{k-1} \end{pmatrix}. \tag{23.6}$$

Im Fall $k = 1$ versteht man unter der Matrix $\mathbf{A}$ die 1×1-Matrix (a_0). Die Matrix $\mathbf{A}$ hängt nur von der linearen Rekursionsrelation ab, die von der linearen Rückkopplungssequenz erfüllt wird.

Lemma [2]:

Ist $s_0, s_1, \ldots$ eine lineare Rückkopplungssequenz, die die Relation (23.5) erfüllt und $\mathbf{A}$ ist die in (23.6) definierte Matrix, dann gilt für die Zustandsvektoren der Sequenz:

$$\vec{s}_n = \vec{s}_0 \cdot \mathbf{A}^n \text{ für } n = 0, 1, 2, \ldots . \tag{23.7}$$

Beispiel [23.82]

Wir betrachten nochmals die Sequenz aus dem letzten Beispiel:

$$s_{n+4} = s_{n+2} + s_n$$

über $\mathbb{F}_2$. Die zu dieser Sequenz assoziierte 4×4 Matrix ist

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Damit erhalten wir:

$$\vec{s}_1 = \vec{s}_0 \cdot \mathbf{A} = (0,0,0,1) \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = (0,0,1,0).$$

$$\vec{s}_2 = \vec{s}_1 \cdot \mathbf{A} = (0,0,1,0) \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = (0,1,0,1).$$

$$\vec{s}_3 = \vec{s}_2 \cdot \mathbf{A} = (0,1,0,1) \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = (1,0,1,0).$$

$$\vec{s}_4 = \vec{s}_3 \cdot \mathbf{A} = (1,0,1,0) \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = (0,1,0,0).$$

$$\vec{s}_5 = \vec{s}_4 \cdot \mathbf{A} = (0,1,0,0) \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = (1,0,0,0).$$

$$\vec{s}_6 = \vec{s}_5 \cdot \mathbf{A} = (1,0,0,0) \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = (0,0,0,1) = \vec{s}_0.$$

Berechnet man die Potenzen der assoziierten Matrix, dann erhält man:

$$\mathbf{A}^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad \mathbf{A}^3 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix},$$

$$\mathbf{A}^4 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad \mathbf{A}^5 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

und:

$$\mathbf{A}^6 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{1}_{4\times 4}.$$

Beispiel [23.83]

Um zu sehen, wie mit Hilfe von Matrizen die Sequenzen generiert werden, betrachten wir ein einfaches Beispiel. Sei

$$s_0, s_1, s_2, \ldots$$

eine lineare, homogene Rückkopplungssequenz über einem endlichen Körper $\mathbf{F}_q$ mit der Relation

$$s_{n+4} = a_3 s_{n+3} + a_2 s_{n+2} + a_1 s_{n+1} + a_0 s_n.$$

Ein beliebiger Zustandsvektor hat die Form

$$\vec{s}_n = (s_n, s_{n+1}, s_{n+2}, s_{n+3})$$

und die Matrix (23.6) hat die Form:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & a_0 \\ 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \end{pmatrix}.$$

Dann ist:

$$\begin{aligned} \vec{s}_n \cdot \mathbf{A} &= (s_n, s_{n+1}, s_{n+2}, s_{n+3}) \cdot \begin{pmatrix} 0 & 0 & 0 & a_0 \\ 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \end{pmatrix} \\ &= (s_{n+1}, s_{n+2}, s_{n+3}, a_0 s_n + a_1 s_{n+1} + a_2 s_{n+2} + a_3 s_{n+3}) \\ &= (s_{n+1}, s_{n+2}, s_{n+3}, s_{n+4}), \end{aligned}$$

wobei im letzten Schritt die Rückkopplungsrelation verwendet wird.

Nun ist:

$$\mathbf{A}^2 = \begin{pmatrix} 0 & 0 & 0 & a_0 \\ 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & a_0 \\ 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & a_0 & a_0 a_3 \\ 0 & 0 & a_1 & a_0 + a_1 a_3 \\ 1 & 0 & a_2 & a_1 + a_2 a_3 \\ 0 & 1 & a_3 & a_2 + a_3 a_3 \end{pmatrix}.$$

Dann ist einerseits:

$$\begin{aligned}\vec{s}_{n+1}\cdot\mathbf{A} &= (s_{n+1},s_{n+2},s_{n+3},s_{n+4})\cdot\begin{pmatrix}0&0&0&a_0\\1&0&0&a_1\\0&1&0&a_2\\0&0&1&a_3\end{pmatrix}\\ &= (s_{n+2},s_{n+3},s_{n+4},a_0s_{n+1}+a_1s_{n+2}+a_2s_{n+3}+a_3s_{n+4})\\ &= (s_{n+2},s_{n+3},s_{n+4},s_{n+5})\end{aligned}$$

mit

$$s_{n+5} = a_0s_{n+1}+a_1s_{n+2}+a_2s_{n+3}+a_3s_{n+4}.$$

Andererseits:

$$\begin{aligned}\vec{s}_n\cdot\mathbf{A}^2 &= (s_n,s_{n+1},s_{n+2},s_{n+3})\cdot\begin{pmatrix}0&0&a_0&a_0a_3\\0&0&a_1&a_0+a_1a_3\\1&0&a_2&a_1+a_2a_3\\0&1&a_3&a_2+a_3a_3\end{pmatrix}\\ &= \big(s_{n+2},s_{n+3},as_n+a_1s_{n+1}+a_2s_{n+2}+a_3s_{n+3},\\ &\qquad a_3(a_0s_n+a_1s_{n+1}+a_2s_{n+2}+a_3s_{n+3})+\\ &\qquad a_0s_{n+1}+a_1s_{n+2}+a_2s_{n+3}\big)\\ &= (s_{n+2},s_{n+3},s_{n+4},s_{n+5}).\end{aligned}$$

Der Formalismus ist daher konsistent.

Die Menge der nichtsingulären $k\times k$-Matrizen über $\mathbb{F}_q$ bilden eine endliche Gruppe unter der Matrizenmultiplikation, die **allgemeine lineare Gruppe** $\mathtt{GL}(k,\mathbf{F}_q)$.

Theorem [19]:

Sei $s_0,s_1,s_2,\ldots$ eine lineare homogene Rückkopplungssequenz der Ordnung k in $\mathbb{F}_q$, die die Relation

$$s_{n+k} = a_{k-1}s_{n+k-1}+a_{k-2}s_{n+k-2}+\ldots+a_0s_n,\quad n=0,1,\ldots$$

erfüllt mit $a_0\neq 0$. Dann ist die kleinste Periode der Folge ein Teiler der Ordnung der assoziierten Matrix $\mathbf{A}$ aus Gl. (23.6) in der allgemeinen linearen Gruppe $\mathtt{GL}(k,\mathbf{F}_q)$.

Beweis:

Die assoziierte Matrix $\mathbf{A}$ hat die Form:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & a_0 \\ 1 & 0 & 0 & \cdots & 0 & a_1 \\ 0 & 1 & 0 & \cdots & 0 & a_2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & a_{k-1} \end{pmatrix}$$

Die Determinante dieser Matrix erhält man durch Entwicklung nach der ersten Zeile zu:

$$\det(\mathbf{A}) = (-1)^{k-1} a_0 \neq 0$$

i.e. $\mathbf{A}$ ist eine nichtsinguläre $k \times k$-Matrix über $\mathtt{GL}(k, \mathbb{F}_q)$ und daher ein Element aus $\mathtt{GL}(k, \mathbb{F}_q)$. Falls m die Ordnung von $\mathbf{A}$ in $\mathtt{GL}(k, \mathbb{F}_q)$ ist, i.e.

$$\mathbf{A}^m = \mathbf{1}_{k \times k},$$

dann folgt

$$\begin{aligned} \vec{s}_{n+m} &= \vec{s}_0 \cdot \mathbf{A}^{n+m} \\ &= \vec{s}_0 \cdot \mathbf{A}^n \\ &= \vec{s}_n, \quad \forall\, n \geq 0. \end{aligned}$$

Daher ist m eine Periode der linearen Rückkopplungssequenz. Da jede Periode einer Folge von der kleinsten Periode geteilt wird (Lemma [1]), folgt die Aussage.

23.2 Impuls Response Sequenzen, Charakteristische Polynome

Unter allen homogenen linearen Rückkopplungssequenzen in $\mathbf{F}_q$, die eine gegebene lineare Rückkopplungsrelation wie Gl. (23.5) erfüllen, können wir diejenige herausheben, die den maximalen Wert für die kleinste Periode in der entsprechenden Kalsse von Folgen annimmt. Diese Folge nennt man **Impuls Response Sequenz** $d_0, d_1, \ldots$, die durch die Startwerte

$$d_0 = d_1 = \cdots = d_{k-2} = 0, d_{k-1} = 1$$

und die lineare Rückkopplungsrelation

$$d_{n+k} = a_{k-1}d_{n+k-1} + a_{k-2}d_{n+k-2} + \cdots + a_0 d_n, n = 0, 1, 2, \ldots \tag{23.8}$$

eindeutig bestimmt ist.

Beispiel [23.84]

Wir betrachten die lineare Rückkopplungsrelation

$$s_{n+5} = s_{n+1} + s_n, \quad n = 0, 1, \ldots \quad \text{in } \mathbf{F}_2. \tag{23.9}$$

Die Startwerte der Folge sind:

$$d_0 = d_1 = d_2 = d_3 = 0, d_4 = 1.$$

Die Impuls Response Sequenz, die mit dieser Rückkopplungsrelation verbunden ist, lautet:

$$0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1. \tag{23.10}$$

Diese Sequenz hat die kleinste Periode 21.

$$\begin{array}{ll}
d_0 = 0 & d_{13} = d_9 + d_8 = 0 \\
d_1 = 0 & d_{14} = d_{10} + d_9 = 1 \\
d_2 = 0 & d_{15} = d_{11} + d_{10} = 0 \\
d_3 = 0 & d_{16} = d_{12} + d_{11} = 1 \\
d_4 = 1 & d_{17} = d_{13} + d_{12} = 1 \\
d_5 = d_1 + d_0 = 0 & d_{18} = d_{14} + d_{13} = 1 \\
d_6 = d_2 + d_1 = 0 & d_{19} = d_{15} + d_{14} = 1 \\
d_7 = d_3 + d_2 = 0 & d_{20} = d_{16} + d_{15} = 1 \\
d_8 = d_4 + d_3 = 1 & d_{21} = d_{17} + d_{16} = 0 \\
d_9 = d_5 + d_4 = 1 & d_{22} = d_{18} + d_{19} = 0 \\
d_{10} = d_6 + d_5 = 0 & d_{23} = d_{19} + d_{18} = 0 \\
d_{11} = d_7 + d_6 = 0 & d_{24} = d_{20} + d_{19} = 0 \\
d_{12} = d_8 + d_7 = 1 & d_{25} = d_{21} + d_{20} = 1
\end{array}$$

Hieraus erhalten wir die Zustandsvektoren:

$$\begin{aligned}
\vec{s}_0 &= (d_0, d_1, d_2, d_3, d_4) = (0,0,0,0,1)\\
\vec{s}_1 &= (d_1, d_2, d_3, d_4, d_5) = (0,0,0,1,0)\\
\vec{s}_2 &= (d_2, d_3, d_4, d_5, d_6) = (0,0,1,0,0)\\
\vec{s}_3 &= (d_3, d_4, d_5, d_6, d_7) = (0,1,0,0,0)\\
\vec{s}_4 &= (d_4, d_5, d_6, d_7, d_8) = (1,0,0,0,1)\\
\vec{s}_5 &= (d_5, d_6, d_7, d_8, d_9) = (0,0,0,1,1)\\
\vec{s}_6 &= (d_6, d_7, d_8, d_9, d_{10}) = (0,0,1,1,0)\\
\vec{s}_7 &= (d_7, d_8, d_9, d_{10}, d_{11}) = (0,1,1,0,0)\\
\vec{s}_8 &= (d_8, d_9, d_{10}, d_{11}, d_{12}) = (1,1,0,0,1)\\
\vec{s}_9 &= (d_9, d_{10}, d_{11}, d_{12}, d_{13}) = (1,0,0,1,0)\\
\vec{s}_{10} &= (d_{10}, d_{11}, d_{12}, d_{13}, d_{14}) = (0,0,1,0,1)\\
\vec{s}_{11} &= (d_{11}, d_{12}, d_{13}, d_{14}, d_{15}) = (0,1,0,1,0)\\
\vec{s}_{12} &= (d_{12}, d_{13}, d_{14}, d_{15}, d_{16}) = (1,0,1,0,1)\\
\vec{s}_{13} &= (d_{13}, d_{14}, d_{15}, d_{16}, d_{17}) = (0,1,0,1,1)\\
\vec{s}_{14} &= (d_{14}, d_{15}, d_{16}, d_{17}, d_{18}) = (1,0,1,1,1)\\
\vec{s}_{15} &= (d_{15}, d_{16}, d_{17}, d_{18}, d_{19}) = (0,1,1,1,1)\\
\vec{s}_{16} &= (d_{16}, d_{17}, d_{18}, d_{19}, d_{20}) = (1,1,1,1,1)\\
\vec{s}_{17} &= (d_{17}, d_{18}, d_{19}, d_{20}, d_{21}) = (1,1,1,1,0)\\
\vec{s}_{18} &= (d_{18}, d_{19}, d_{20}, d_{21}, d_{22}) = (1,1,1,0,0)\\
\vec{s}_{19} &= (d_{19}, d_{20}, d_{21}, d_{22}, d_{23}) = (1,1,0,0,0)\\
\vec{s}_{20} &= (d_{20}, d_{21}, d_{22}, d_{23}, d_{24}) = (1,0,0,0,0)\\
\vec{s}_{21} &= (d_{21}, d_{22}, d_{23}, d_{24}, d_{25}) = (0,0,0,0,1) = \vec{s}_0.
\end{aligned}$$

Die zu der Sequenz (23.9) assoziierte Matrix ist:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Dann sind die Potenzen dieser Matrix:

$$\mathbf{A}^2 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{A}^3 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

$$\mathbf{A}^4 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{A}^5 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

$$\mathbf{A}^6 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \mathbf{A}^7 = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

$$\mathbf{A}^8 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \mathbf{A}^9 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

$$\mathbf{A}^{10} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}, \mathbf{A}^{11} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{A}^{12} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}, \mathbf{A}^{13} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

$$\mathbf{A}^{14} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \mathbf{A}^{15} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

$$\mathbf{A}^{16} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \mathbf{A}^{17} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

$$\mathbf{A}^{18} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \mathbf{A}^{19} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{A}^{20} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{A}^{21} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{1}_{5\times 5}.$$

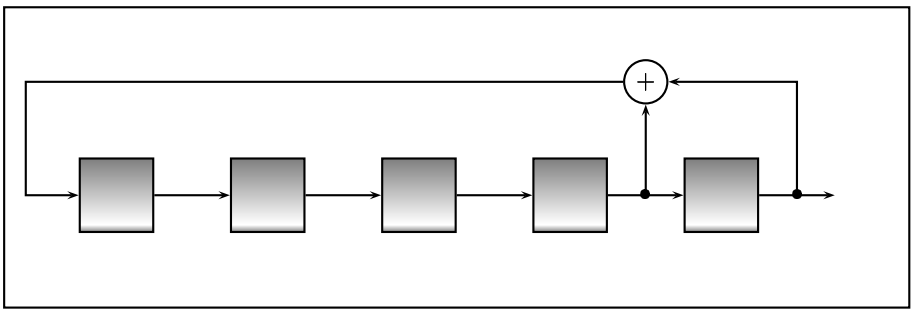


Abbildung 23.6: Lineares Feedback Shift Register der Sequenz (23.9)

Ein Feedback Shift Register, das die Folge (23.10) generiert, ist in der Abbildung [23.6] dargestellt. Man kann sich vorstellen, dass die Folge (23.9) dadurch erzeugt wird, dass zu Beginn der Verarbeitung alle Delay Elemente den Startwert 0 haben und den 'Impuls' 1 in das Delay Element ganz links setzt. Dies erklärt die Terminologie *Impuls Response Sequenz*.

Lemma [3]:

Sei $d_0, d_1, \ldots$ die Impuls Response Sequenz in $\mathbb{F}_q$, die die Relation (23.8) erfüllt und sei $\mathbb{A}$ die Matrix (23.6). Zwei Zustandsvektoren $\vec{d}_m$ und $\vec{d}_n$ sind dann und nur dann gleich, wenn

$$\mathbf{A}^m = \mathbf{A}^n.$$

Beweis:

$\Longrightarrow$ Die hinreichende Bedingung folgt aus Lemma [2].

$\Longleftarrow$ Sei $\vec{d}_m = \vec{d}_n$. Aus der linearen Rekursionsrelation (23.8) folgt dann

$$\vec{d}_{m+t} = \vec{d}_{n+t} \quad \forall\, t \geq 0.$$

Mit dem Lemma [2] erhalten wir

$$\vec{d}_t \cdot \mathbf{A}^m = \vec{d}_t \cdot \mathbb{A}^n \quad \forall t \geq 0.$$

Da die Vektoren $\vec{d}_0, \vec{d}_1, \ldots, \vec{d}_{k-1}$ offensichtlich ein Basissystem des Vektorraums $\mathbb{F}_q^k$ über $\mathbb{F}_q$ aufspannen, folgt $\mathbf{A}^m = \mathbf{A}^n$.

Theorem [20]:

Die kleinste Periode eine linearen Rekursionssequenz in $\mathbb{F}_q$ teilt die kleinste Periode der korrespondierenden Impuls Response Sequenz.

Beweis:

Theorem [21]:

Ist $d_0, d_1, \ldots$ eine Impuls Response Sequenz der Ordnung k in $\mathbb{F}_q$, die die Relation

$$d_{n+k} = a_{k-1}d_{n+k-1} + a_{k-2}d_{n+k-2} + \cdots + a_0 d_n, \quad n = 0, 1, 2, \ldots$$

erfüllt mit $a_0 \neq 0$ und $\mathbf{A}$ ist die dazu assoziierte Matrix, dann ist die kleinste Periode der Sequenz gleich der Ordnung von $\mathbf{A}$ in $\mathtt{GL}(k, \mathbb{F}_q)$.

Beweis:

Sei r die kleinste Periode der Sequenz $d_0, d_1, \ldots$. Gemäß Theorem [19] ist r Teiler der Ordnung der Matrix $\mathbf{A}$. Andererseits ist $a_0 \neq 0$, dann folgt aus Theorem [18], dass die Folge $d_0, d_1, \ldots$ periodisch ist, i.e.

$$\vec{d}_r = \vec{d}_0.$$

Dann resultiert aber aus Lemma [3] $\mathbf{A}^r = \mathbf{A}^0$, was das gewünschte Resultat ist.

Beispiel [23.85]

Wir greifen nochmal das Beispiel [23.84] auf. Dort haben wir gezeigt, dass die lineare Rekursionsrelation

$$s_{n+5} = s_{n+1} + s_n, \quad n = 0, 1, 2, \ldots \tag{23.11}$$

in $\mathbb{F}_2$ die kleinste Periode 21 für die korrespondierende Impuls Response Sequenz

hat. Die assoziierte Matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

hat die Ordnung 21 in $\mathtt{GL}(5, \mathbb{F}_2)$. Die lineare Rekursionsrelation (23.11) hat $2^5 = 32$ verschiedene Startvektoren. Ist der Startvektor einer linearen Rekursionssequenz gleich einem der 21 verschiedenen Zustandsvektoren, die in der Impuls Response Sequenz auftreten, dann ist die kleinste Periode wieder 21, da solch eine Sequenz nichts anderes darstellt als eine verschobene Impuls Response Sequenz.

Starten wir mit dem Initial Zustandsvektor

$$\vec{s}_0 = (1, 1, 1, 0, 1),$$

dann generiert die Relation (23.11) die Folge:

$$\begin{aligned} s_0 = 1; s_1 = 1; & s_2 = 1; s_3 = 0; s_4 = 1 \\ s_5 = 0; s_6 = 0; & s_7 = 1; s_8 = 1; s_9 = 1 \\ s_{10} = 0; s_{11} = 1; & \cdots \end{aligned}$$

mit den Zustandsvektoren:

$$\begin{aligned} \vec{s}_0 &= (s_0, s_1, s_2, s_3, s_4) = (1, 1, 1, 0, 1) \\ \vec{s}_1 &= (s_1, s_2, s_3, s_4, s_5) = (1, 1, 0, 1, 0) \\ \vec{s}_2 &= (s_2, s_3, s_4, s_5, s_6) = (1, 0, 1, 0, 0) \\ \vec{s}_3 &= (s_3, s_4, s_5, s_6, s_7) = (0, 1, 0, 0, 1) \\ \vec{s}_4 &= (s_4, s_5, s_6, s_7, s_8) = (1, 0, 0, 1, 1) \\ \vec{s}_5 &= (s_5, s_6, s_7, s_8, s_9) = (0, 0, 1, 1, 1) \\ \vec{s}_6 &= (s_6, s_7, s_8, s_9, s_{10}) = (0, 1, 1, 1, 0) \\ \vec{s}_7 &= (s_7, s_8, s_9, s_{10}, s_{11}) = (1, 1, 1, 0, 1) \end{aligned}$$

Diese Sequenz hat die kleinste Periode 7. Wie im obigen Fall ergibt sich die gleiche kleinste Periode, wenn man einen der 7 Zustandsvektoren als Startzustand wählt.

Startet man mit den Zustandsvektor

$$\vec{s}_0 = (1, 1, 0, 1, 1),$$

dann erhält man die Folge:

$$1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ \ldots$$

mit der kleinsten Periode 3. Die gleiche kleinste Periode ergibt sich, wenn man mit einem der drei Zustandsvektoren dieser Folge beginnt. Schließlich

bleibt noch der Startvektor $(0,0,0,0,0)$, der eine Sequenz mit kleinster Periode 1 generiert.

Damit haben wir alle 32 mögliche Startvektoren untersucht.

Theorem [22]:

Sei $s_0, s_1, \ldots$ eine lineare Rekursionssequenz der Ordnung k über $\mathbb{F}_q$ mit Vorperiode n_0. Falls es k Zustandsvektoren

$$\vec{s}_{m_1}, \vec{s}_{m_2}, \ldots, \vec{s}_{m_k}$$

gibt mit $m_j \geq n_0$ für $1 \leq j \leq k$, die linear unabhängig über $\mathbb{F}_q$ sind, dann sind $s_0, s_1, \ldots$ und die korrespondierende Impuls Response Sequenz periodisch und beide haben die gleiche kleinste Periode.

Beweis:

Sei r die kleinste Periode der Sequenz $s_0, s_1, \ldots$. Für $1 \leq j \leq k$ haben wir

$$\vec{s}_{m_j} \mathbf{A}^r = \vec{s}_{m_j+r} = \vec{s}_{m_j}$$

wegen Lemma [2]. Daher ist $\mathbf{A}^r$ die $k \times k$-Einheitsmatrix über $\mathbb{F}_q$. Damit erhalten wir

$$\vec{s}_r = \vec{s}_0 \cdot \mathbf{A}^r = \vec{s}_0,$$

damit ist $s_0, s_1, \ldots$ periodisch.

Wir bezeichnen mit $\vec{d}_n$ den n-ten Zustandsvektor der Impuls Response Sequenz, dann ist

$$\vec{d}_r = \vec{d}_0 \mathbf{A}^r = \vec{d}_0$$

und Theorem [20] vervollständigt den Beweis.

Beispiel [23.86]

Die Bedingung $m_j \geq n_0$ im Theorem [22] ist notwendig, denn es gibt lineare Rekursionssequenzen der Ordnung k, die nicht periodisch sind, jedoch k linear unabhängige Zustandsvektoren haben. Sei $d_0, d_1, \ldots$ die lineare Rekursionssequenz zweiter Ordnung über $\mathbb{F}_q$ mit

$$d_{n+2} = d_{n+1}, n = 0, 1, 2, \ldots$$

mit den Termen

$$0\ 1\ 1\ 1\ \ldots$$

Die Zustandsvektoren $\vec{d}_0, \vec{d}_1$ sind sicherlich linear unabhängig über $\mathbb{F}_q$, aber die Folge ist nicht periodisch. In diesem Fall ist $n_0 = 1$.

Die Umkehrung des Theorems [22] ist falsch, i.e. wenn eine Folge periodisch ist, müssen die Zustandsvektoren nicht linear abhängig sein. Betrachte dazu die lineare Rekursionssequenz dritter Ordnung $s_0, s_1, \ldots$ über $\mathbb{F}_2$ mit

$$s_{n+3} = s_n, n = 0, 1, 2, \ldots$$

und $\vec{s_0} = (1, 1, 0)$. Dann sind sowohl $s_0, s_1, \ldots$ als auch die korrespondierende Impuls Response Sequenz periodisch mit kleinster Periode 3, aber drei beliebige Zustandsvektoren der Folge $s_0, s_1, \ldots$ sind linear abhängig über $\mathbb{F}_2$.

Sei $s_0, s_1, \ldots$ eine lineare Rekursionssequenz der Ordnung k in $\mathbb{F}_q$, die die lineare Rekursionsrelation

$$s_{n+k} = a_{k-1}s_{n+k-1} + a_{k-2}s_{n+k-2} + \ldots a_0 s_n, \quad n = 0, 1, 2, \ldots \tag{23.12}$$

erfüllt, $a_j \in \mathbb{F}_q, 0 \leq j \leq k-1$. Das Polynom

$$f(x) = x^k - a_{k-1}x^{k-1} - a_{k-2}x^{k-2} - \ldots - a_1 x - a_0 \in \mathbb{F}_q[x] \tag{23.13}$$

heißt **charakteristisches Polynom** der linearen Rekursionssequenz. Das charakteristische Polynom (23.13) hängt nur von der linearen Rekursionsrelation (23.12) ab. Ist $\mathbf{A}$ die in (23.6) definierte Matrix, dann ist das Polynom $f(x)$ identisch mit dem charakteristischen Polynom im Sinne der linearen Algebra, i.e.

$$f(x) = \det\left(x \cdot \mathbf{1}_{k\times k} - \mathbf{A}\right).$$

Als erste Anwendung des charakteristischen Polynoms zeigen wir, wie die Terme einer linearen Rekursionssequenz in einem wichtigen Spezialfall explizit dargestellt werden können.

Theorem [23]:

Sei $s_0, s_1, \ldots$ eine homogene, lineare Rekursionssequenz in $\mathbb{F}_q$ mit charakteristischem Polynom $f(x)$. Sind die Wurzeln $\alpha_1, \alpha_2, \ldots, \alpha_k$ des Polynoms $f(x)$ alle verschieden, dann ist

$$s_n = \sum_{j=1}^{k} \beta_j \alpha_j^n, \quad n = 0, 1, 2, \ldots, \tag{23.14}$$

wobei die $\beta_1, \beta_2, \ldots, \beta_k$ Elemente sind, die eindeutig durch die Startwerte der Folge bestimmt sind und zu dem Zerfällungskörper von $f(x)$ über $\mathbb{F}_q$ gehören.

Beweis:

Die Konstanten $\beta_1, \beta_2, \ldots, \beta_k$ können aus dem System linearer Gleichungen

$$\sum_{j=1}^{k} \alpha_j^n \beta_j = s_n, \quad n = 0, 1, 2, \ldots, k-1$$

bestimmt werden. Die Determinante dieses Gleichungssystems ist eine VANDERMONDE Determinante, die wegen der Bedingungen an die $\alpha_1, \ldots, \alpha_k$ nicht verschwindet. Daher sind die Elemente $\beta_1, \beta_1, \ldots, \beta_k$ eindeutig bestimmt und sind Elemente des Zerfällungskörpers[1] von $f(x)$ über $\mathbb{F}_q$.

Um die Identität (23.14) für alle $n \geq 0$ zu zeigen, genügt es nachzuweisen, dass die rechte Seite von (23.14) mit den speziellen Werten für $\beta_1, \beta_2, \ldots, \beta_k$ die lineare Rekursionsrelation (23.12) erfüllt. Nun ist:

$$\sum_{j=1}^{k} \beta_j \alpha_j^{n+k} - a_{k-1} \sum_{j=1}^{k} \beta_j \alpha_j^{n+k-1} - \ldots - a_0 \sum_{j=1}^{k} \beta_j \alpha_j^n = \sum_{j=1}^{k} \beta_j f(\alpha_j) \alpha_j^n$$
$$= 0$$

für alle $n \geq 0$.

[1] Engl.: *Splitting field.*

Anhang A

Lösungen zu den Übungen aus Kapitel [14.1.1]

A.1 Übung [14.1]

Seien $m, n \in \mathbb{Z}$ und $k \in \mathbb{N}$. Sind die folgenden Aussagen wahr oder falsch? Begründen Sie Ihre Antwort.

(a) Wenn k ein Teiler von $m + n$ ist, dann auch jeweils von n und m.

(b) Wenn k ein Teiler von $m \cdot n$ ist, dann auch jeweils von n und m.

(c) Wenn k ein Teiler von $m \cdot n$ ist, dann auch von n oder von m.

(d) Wenn k zwar m teilt aber nicht n, dann ist k auch kein Teiler von $n + m$.

(e) Wenn k zwar m teilt aber nicht n, dann ist k auch kein Teiler von $n \cdot m$.

(f) Wenn k jeweils n und m mit Rest 1 teilt, dann teilt k auch $n \cdot m$ mit Rest 1.

(g) Wenn k jeweils n und m mit Rest 1 teilt, dann teilt k auch $n + m$ mit Rest 1.

Lösung:

(a) Die Aussage

> Wenn k ein Teiler von $m + n$ ist, dann auch jeweils von n und m

ist falsch. Gegenbeispiel: 2 ist ein Teiler von 8, und $8 = 3 + 5$, aber 2 teilt nicht 3 und nicht 5.

(b) Die Aussage

> Wenn k ein Teiler von $m \cdot n$ ist, dann auch jeweils von n und m

ist falsch. Gegenbeispiel: 6 ist Teiler von 18, aber $18 = 2 \cdot 9$ und 6 ist weder Teiler von 2 noch Teiler von 9.

(c) Die Aussage

> Wenn k ein Teiler von $m \cdot n$ ist, dann auch von n oder von m

ist falsch, Gegenbeispiel wie zuvor.

(d) Die Aussage

> Wenn k zwar m teilt aber nicht n, dann ist k auch kein Teiler von $n + m$

ist wahr. Wir haben die Voraussetzung, dass m durch k teilbar ist, die Zahl n aber nicht.
Wir nehmen an, die Zahl $n + m$ ist ebenfalls durch k teilbar. Dann folgt, es gibt Zahlen $a, b \in \mathbb{Z}$, so dass

$$m + n = a \cdot k \qquad \text{und} \qquad m = b \cdot k.$$

Dann ist:

$$n = (m + n) - m = a \cdot k - b \cdot k = (a - b) \cdot k,$$

und $a - b \in \mathbb{Z}$. Daher ist n durch k ebenfalls teilbar, was ein Widerspruch zur Annahme darstellt.

(e) Die Aussage

> Wenn k zwar m teilt aber nicht n, dann ist k auch kein Teiler von $n \cdot m$

ist falsch. Gegenbeispiel: 3 teilt 6 und nicht 5, aber 3 teilt $6 \cdot 5 = 30$.

(f) Die Aussage

> Wenn k jeweils n und m mit Rest 1 teilt, dann teilt k auch $n \cdot m$ mit Rest 1

ist wahr. Wenn k die Zahlen m und n mit Rest 1 teilt, dann gibt es Zahlen $a, b \in \mathbb{Z}$ mit

$$\begin{aligned} n &= a \cdot k + 1, \\ m &= b \cdot k + 1. \end{aligned}$$

Dann ist

$$\begin{aligned} n \cdot m &= (a \cdot k + 1) \cdot (b \cdot k + 1) \\ &= abk^2 + ak + bk + 1 \\ &= (abk + a + b) \cdot k + 1. \end{aligned}$$

Daher ist auch $n \cdot m$ durch k mit Rest 1 teilbar.

(g) Die Aussage

> Wenn k jeweils n und m mit Rest 1 teilt, dann teilt k auch $n+m$ mit Rest 1

ist falsch. Gegenbeispiel: 3 teilt 4 und 7 jeweils mit Rest 1, aber $4+7 = 11$ mit Rest 2.

Anhang B

Lösungen zu den Übungen aus Kapitel [18.3.1]

B.1 Übung [18.2]

Berechnen Sie $d = \mathtt{ggT}(360, 294)$ auf drei Arten:

1. durch Zerlegung der beiden Zahlen in ihre Primfaktoren und daraus die Primfaktor Zerlegung von d.
2. mit Hilfe des EUKLIDischen Algorithmus.
3. mit Hilfe der STEIN Algorithmus.

Lösung:

B.2 Übung [18.3]

Berechnen Sie für jedes der folgenden Paare von Zahlen den größten gemeinsamen Teiler mit Hilfe des EUKLIDischen Algorithmus. Drücken Sie den ggT dann aus als Linearkombination dieser beiden Zahlen.

1. 26, 19
2. 187, 34
3. 841, 160
4. 2613, 2171

Lösung:

B.3 Übung [18.4]

Berechnen Sie mit Hilfe des erweiterten EUKLIDischen Algorithmus das multiplikative Inverse

$$19^{-1} \bmod 251.$$

Lösung:

B.4 Übung [18.5]

Berechnen Sie das multiplikative Inverse

$$33^{-1} \bmod 742.$$

Lösung:

Anhang C

Akronyme

Im IT Bereich – unter anderem auch in der Kryptographie – ist es sehr verbreitet, mit Abkürzungen zu arbeiten. Daher erscheint es sehr zweckmäßig, hier ein – sicher nicht vollständiges – Abkürzungsverzeichnis anzufügen.

Akronym	**Klartext**
3DES	Triple Data Encryption Standard
ACM	Association for Computing Machinery
AES	Advanced Encryption Standard
AH	Authentication Header
ANSI	American National Standards Institute
ASCII	American Standard Code of Information Interchange
ATM	Automatic Teller Machine
BBS	Blum-Blum-Shub
BSI	Bundesamt für Sicherheit in der Informationstechnologie
CAST	
CBC	Cipher Block Chaining
CERT	Computer Emergency Response Team
CFB	Cipher Feedback
CRC	Cyclic Redundancy Check
CTR	Counter Mode
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECB	Electronoc Code Book
ECC	Error Correcting Code
ECC	Elliptic Curve Cryptography
ECDDHP	Elliptic Curve Decision Diffie Hellman Problem
ECDH	Elliptic Curve Diffie Hellman

Akronym	Klartext
ECDHP	Elliptic Curve Diffie Hellman Problem
ECDLP	Elliptic Curve Discrete Logarithmus Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EFF	Electronic Frontier Foundation
ESP	Encapsulated Securoty Payload
FAQ	Frequently Asked Questions
FEAL	Fast Data Encipherment Algorithm
FIPS-PUB	Federal Information Standards Publication
FSF	Free Software Foundation
GC&CS	Government Code and Cipher School
GCHQ	Government Communications Headquaters
ggT	Größter Gemeinsamer Teiler
HDLC	High Level Data Link Protocol
HMAC	Hash based Message Authentication Code
HTTP	Hypertext Transfer Protocol
IAB	Internet Architecture Board
IDEA	International Data Encryption Algorithm
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IFF	Identification Friend or Foe
IP	Initial Permutation
ISO	International Organisation for Standardization
ITU	International Telecommunication Union
IV	Initial Vector
LFSR	Linear Feedback Shift Register
MAC	Message Authentication Code
MD5	Message Digest, Version 5
MIME	Multipurpose Internet Mail Extension
MIT	Massachusetts Institute of Technology
NBS	National Bureau of Standards
NESSIE	New European Schemes for Signature Integrity, and Encryption
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OFB	Output Feedback
OSI	Open Systems Interconnectivity
PGP	Pretty Good Privacy

Akronym	Klartext
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PRNG	Pseudo Random Number Generator
RFC	Request for Comments
RIPE	Reseaux IP Europe'ens
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
SNMP	Simple Network Management Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VPN	Virtual Private Network
WEP	Wireless Equivalent Privacy
WLAN	Wireless Local Area Network
WPA	

Anhang D

Glossar

Abelsche Gruppe

Eine ABELsche Gruppe ist eine abstrakte Gruppe mit einer kommutativen, binären Operation.

AES

Der *Advanced Encryption Standard* (AES) ist das symmetrische Verschlüsselungsverfahren, das den Data Encryption Standard als Standardverfahren der symmetrischen Verschlüsselung abgelöst hat.

Asymmetrische Verschlüsselung

Unter asyymetrischer Verschlüsselung versteht man kryptographische Verfahren, die zur Ver– und Entschlüsselung zwei verschiedene, mathematisch auf subtile Weise voneinander abhängige Schlüssel, den öffentlichen Public-Key und den geheimen Private-Key, einsetzen.

Authentifikation

Unter dem Begriff *Authentifikation* versteht man einen Prozess, der bestimmte Informationen verifiziert beziehungsweise die Identität eines Subjektes beweist.

Autorisierung

Autorisierung bedeutet die Zuweisung von Rechten zu einem Subjekt.

Avalanche Effekt

Der Avalanche Effekt dadurch charakterisiert, daß in einem Verschlüsselungsalgorithmus eine kleine Änderung des Klartextes oder des Schlüssels eine große Änderung im Chiffretext bewirkt.

Block Chiffre

Eine Block Chiffre bezeichnet ein symmetrisches oder asyymetrisches Verschlüsselungsverfahren, das Klartextblöcke bestimmter Länge (zum Beispiel 64 Bit) in einem Vorgang auf einen Chiffretextblock abbildet.

Brute-Force Attacke

Unter *Brute-Force Attacke* versteht man eine Angriffsart gegen ein Kryptosystem, bei dem systematisch alle möglichen Schlüsselwerte ausprobiert werden.

Chiffretext

Der Chiffretext ist der Output eines Verschlüsselungsalgorithmus, also die verschlüsselte Form einer Nachricht.

Codierung

Diffusion

Unter Diffusion versteht man eine kryptographische Technik, deren Ziel die statistische Häufigkeit des Klartexts bei der Verschlüsselung zu verschleieren. Dies wird dadurch erreicht, dass durch den Chiffrieralgorithmus möglichst viele Chiffretextzeichen von einem Klartextzeichen abhängen.

Digitale Signatur

Unter Digitaler Signatur versteht man ein Authentifikationsmechanismus. Diese setzt den Erzeuger einer Nachricht in die Lage, einen Code an eine Nachricht anzuhängen. Dieser Code stellt die Signatur dar. Die Signatur wird gebildet, indem der Hashwert der Nachricht erzeugt wird. Dieser Hashwert wird mit dem privaten Schlüssel des Senders verschlüsselt. Digitale Signatur beweist die Identität des Senders eindeutig, gleichzeitig wird dadurch die Integrität realisiert.

Digramm

Unter Digramm versteht man eine Folge von zwei Buchstaben. In der deutschen, englischen oder anderen Sprachen kann die relative Häufigket des Auftretens von Digrammen im Klartext in der Kryptoanalyse verschiedener Chiffren benutzt werden.

Diskretes Logarithmus Problem

Digital Signature Algorithm (DSA)

Digital Signature Standard (DSS)

Erweiterter Euklidischer Algorithmus

Der erweiterte EUKLIDische Algorithmus ist ein systematisches verfahren zur Berechnung des multiplikativen Inversen (mod n).

Euklidischer Algorithmus

Ein systematisches Verfahren, das in polynomialer Laufzeit den größten gemeinsamen Teiler (ggT) zweier Zahlen berechnet.

Eulersche Totientenfunktion

Faktorisierung

Unter Faktorisierung versteht man die Zerlegung einer gegebenen natürlichen Zahl in das Produkt von Potenzen der Primfaktoren.

Feistel Netzwerk

Unter FEISTEL Netzwerk – oder auch FEISTEL Chiffre genannt – versteht man eine spezielle Klasse von iterierten Block Chiffren, bei denen der Chiffretext aus der wiederholten Anwendung der gleichen Funktion – der *Rundenfunktion* – berechnet wird.

Gruppe

Eine *Gruppe* ist eine algebraische Struktur mit assoziativer binärer Verknüpfung, Einselement und inversem Element.

Hash Funktion

Eine *Hash-Funktion* bildet einen Input variabler Länge – e.g. eine Nachricht – auf einen Outpt ab mit fester Länge.

Identifikation

Integrität

Die Integrität einer Nachricht bedeutet, daß der Inhalt einer Nachricht während der Übertragung über einen unsicheren Kommunikationskanal von nicht autorisierten Personen nicht geändert werden kann.

Kerckhoffs Prinzip

Das KERCKHOFFS Prinzip besagt, dass die Sicherheit eines Kryptosystems nicht von der Geheimhaltung des Verschlüsselungsalgorithmus abhängen darf. Die Sicherheit eines Kryptosystems muß von der Geheimhaltung eines Schlüssels abhängen.

Klartext

Klartext bezeichnet den Input eines Verschlüsselungsalgorithmus oder den Output eines Dechiffierverfahrens.

Konfusion

Unter Konfusion versteht man eine kryptographische Technik, deren Ziel die Beziehimg zwischen der Statistik des Klartexts und dem Wert des Schlüssels so komplex wie möglich zu machen.

Kryptosystem

MD4

MD4 ist eine One-Way Hash Funktion, die von Ron Rivest entwickelt wurde [193, pp. 435–436].

MD5

MD5 ist eine verbesserte Version der MD4 Hash Funktion. MD5 ist komplexer als MD4, jedoch ähnlich designed wie MD4 und produziert einen 128 Bit Message Digest. Siehe [193, pp. 436–441].

Message Authentication Code (MAC)

Der *Message Authentication Code* ist eine kryptographische Prüfsumme.

Message Digest

Modulare Arithmetik

Monoalphabetische Chiffre

Unter einer *monoalphabetischen Chiffre* versteht man ein Verschlüsselungsverfahren, bei dem ein Klartextzeichen immer auf das gleiche Chiffretextzeichen abgebildet wird. Anders formuliert: Das Klartextalphabet wird auf genau ein Chiffretextalphabet abgebildet.

Nonce

Unter dem Begriff *Nonce* versteht man in der Kryptographie einen Identifizierer oder eine Zahl, die genau ein Mal verwendet werden darf.

NSA

Die *National Security Agency* (NSA) ist eine amerikanische Regierungsbehörde, deren Aufgabe die Entschlüsselung und Überwachung von ausländischer (gelegentlich auch inländischer) Kommunikation ist.

One-Way Hash Funktion

Passwort

In der Kryptographie versteht man unter *Passwort* eine Zeichenkette, die eine Identität authentifiziert. Die Kenntnis des Passworts und der zugehörigen UserID wird als Beweis angesehen, um die Autorisierung der UserID freizugeben.

Permutation

Polyalphabetische Chiffre

Unter einer *polyalphabetischen Chiffre* versteht man ein Verschlüsselungsverfahren, bei dem ein Klartextzeichen auf verschiedene Chiffretextzeichen abgebildet wird. Anders formuliert: Das Klartextalphabet wird auf mehrere Chiffretextalphabete abgebildet.

Primitive Wurzel

Primzahl

Eine Primzahl ist eine positive ganze Zahl p, deren einzigen Teiler die Zahlen ± 1 und $\pm p$ sind.

Pseudozufallszahl

Darunter versteht man eine Zahl – bzw. Folge von Zahlen – die durch einen deterministischen Algorithmus erzgt wird. Die Zahlenfolge erscheint zufällig.

Pseudozufallszahlengenerator

Ein Pseudozufallszahlengenerator eist eine Funktion, die deterministisch eine folge von Zahlen generiert, die statistisch zufällig ist.

RIPE

RSA Algorithmus

Der RSA Algorithmus ist ein Public-Key Verfahren, das auf der Exponentiation in der modularen Arithmetik basiert.

Schlüsselraum

Der Schlüsselraum bezeichnet die Menge der möglichen Schlüssel eines Kryptosystems.

Session Key

Der Session Key oder Sitzungsschlüssel ist ein temprärer Schlüssel zum Ver-/Entschlüsseln von Nachrichten zwischen zwei Parteien.

Steganographie

Unter Steganographie versteht man Verfahren, die *Existenz* – nicht deren Inhalt – einer Nachricht zu verschleieren.

Stromchiffre

Stromchiffren sind symmetrische oder asymmetrische Verschlüsselungsverfahren, die den Chiffretext zeichenweise aus dem Klartext generieren.

Substitution

Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung bezeichnet Kryptosysteme, die zur Ver- bzw. Entschlüsselung einen Schlüssel verwenden. Dieser Schlüssel muß über einen sicheren Übertragungskanal zwischen den beiden Kommunikationspartnern ausgetauscht werden.

Traffic-Analysis

Unter *Traffic-Analysis* versteht man

Trap-Door Funktion

Eine Trapdoor Funktion hat die Eigenschaft, dass sie mit rechnerischem Aufwand einfach zu berechnen ist, die Berechnung der Umkehrfunktion ist jedoch nur mit sehr hohem Aufwand zu erhalten.

Uneingeschränkte Sicherheit

Bezeichnet ein Kryptosystem, das sicher ist gegenüber einem Angreifer, der über unbegrenzte Zeit und unbegrenzte Rechenkapazität verfügt.

Virual Private Network (VPN)

Ein *virtual private network* besteht aus einer Reihe von Comoputern, die durch ein unsicheres Netzwerk verbunden sind. Durch Verwendung von Verschlüsselungsmethoden und Protokollen wird Sicherheit realisiert.

Wasserzeichen

Zeitstempel

Literaturverzeichnis

[1] Agrawal M., N. Kayal and N. Saxena
Primes is in $\mathcal{P}$
Annals of Mathematics **160** (2): 781-793, 2004.

[2] Anderson, Ross
Security Engineering
A Guide to Building Dependable Distributed Systems
John Wiley & Sons.
New York, 2001.

[3] Appelgate, David L., Robert E. Bixby, Vasek Chvatal and William J. Cook
the Traveling Salesman Problem
A Computational Study
Princeton University Press
Princeton, New Jersey, 2006.

[4] Arbaugh, William A., Narendar Shankar and Y. C. Justin Wan
Your 802.11 Wireless Network has No Clothes
University of Maryland
March 30, 2001
In: *Proceedings of the First IEEE International Conference on Wireless LANs and Home Networks*
December 2001.
URL: http://www.cs.umd.edu/~waa/papers.shtmlf

[5] Ash, Avner and Robert Gross
Elliptic Tales
Curves, Counting, and Number Theory
Princeton University Press
Princeton, New Jersey, 2012.

[6] Bamford, James
NSA
Die Anatomie des mächtigsten Geheimdienstes der Welt
Wilhelm Goldmann Verlag
München, 2002.

[7] Barrett, Daniel J. and Richard E. Silverman
SSH The Secure Shell
The definite Guide

O'Reilly
Beijing, Cambridge, 2001.

[8] Bashmakova, Isabella Grigoryevna
Diophantus and Diophantine Equations
The Mathematical Association of America, 1997.

[9] Basieux, Pierre
Die Architektur der Mathematik
Denken in Strukturen
Rowohlt Taschenbuch Verlag
Reinbek bei Hamburg, 2000.

[10] Bartolome, Andreas, Josef Rung und Hans Kern
Zahlentheorie für Einsteiger
5. Auflage
Friedrich Vieweg Verlag
Braunschweig/Wiesbaden, 2006.

[11] Bauer, Craig P.
Secret History
The Story of Cryptology
CRC Press, Taylor & Francis, 2013.

[12] Bauer, Craig P.
Unsolved
The History and Mystery of the World's Greatest Ciphers from Ancient Egypt to Online Secret Societies
Princeton University Press, Princeton, NJ, 2017.

[13] Bauer, Friedrich L.
Entzifferte Geheimnisse
2., erweiterte Auflage
Springer Verlag
Berlin, Heidelberg, New York, 1997.

[14] Bauer, Friedrich L.
Erich Hüttenhain: Entzifferung 1939 – 1945
Informatik Spektrum, 4, 2008, 249 – 261.

[15] Beckman, Bengt
Codebreakers
Arne Beurling and the Swedish Crypto Program during World War II
American Mathematical Society
Rhode Islands, 2002.

[16] Bellare, Mihir, Ran Canetti and Hugo Krawczyk
Keying Hash Functions for Message Authentication
Advances in Cryptology – Crypto 96
LNCS 1109, Springer Verlag 1996.

[17] Bellare, Mihir, Ran Canetti and Hugo Krawczyk
Message Authentication using Hash Functions – The HMAC Construction
CryptoBytes **2** (no. 1) (1996)

[18] Bennett, Charles H., and Gilles Brassard
Quantum cryptography: Public Key distribution and coin tossing
in: *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*
pp. 175 – 179, IEEE, New York, 1984. bangalore, India, December 1984.

[19] Bennett, Charles H., and Gilles Brassard
The Dawn of a New Era for Quantum Cryptography: The Experimantal Prototype is working!
SIGACT News, Vol. 20, pp. 78–82 (1989).

[20] Berlekamp, Elwyn R.
Algebraic Coding Theory
McGraw–Hill, New York, 1968.

[21] Bernstein, Daniel J., Johannes Buchmann and Erik Dahmen (Ed.)
Post–Quantum Cryptography
Springer Verlag
Berlin, Heidelberg, 2009.

[22] Beutelspacher, Alfred, Jörg Schwenk, Klaus–Dieter Wolfenstetter
Moderne Verfahren der Kryptographie
Von RSA zu Zero–Knowledge
Friedrich Vieweg Verlag
Braunschweig/Wiesbaden, 2001.

[23] Beutelspacher, Alfred, Heile B. Neumann, Thomas Schwarzpaul
Kryptographie in Theorie und Praxis
Mathematische Grundlagen für elektronisches Geld, Internetsicherheit und Mobilfunk
Friedrich Vieweg Verlag
Braunschweig/Wiesbaden, 2005.

[24] Beyrer, Klaus (Hrg.)
Streng geheim
Die Welt der verschlüsselten Kommunikation
Umschau/Braus
Heidelberg, 1999.

[25] Blake, Ian, Gadiel Seroussi and Nigel Smart
Elliptic Curves in Cryptography
Cambridge University Press
Cambridge, 1999.

[26] Blum, L., M. Blum and M. Shub
A Simple Unpredictable Pseudo-Random Number Generator
SIAM Journal of Computing, Vol. 15, Nr. 2, May 1986, pp. 364 – 383.

[27] Borisov, Nikita, Ian Goldberg and David Wagner
Intercepting Mobile Communications: The Insecurity of 802.11

In: *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking*, pp. 180 – 188, 2001.
URL: http://www.isaac.cs.berkeley.edu/isaac/wep-draft.pdf

[28] Braden, R., D. Clark, S. Crocker and C. Huitema
Report of IAB Workshop on Security in the Internet Architecture
RFC 1636, June 1994.

[29] Brands, Gilbert
Verschlüsselungsalgorithmen
Angewandte Zahlentheorie rund um Sicherheitsprotokolle
Friedrich Vieweg Verlag
Braunschweig/Wiesbaden, 2002.

[30] Brands, Stefan
Untraceable Off–line Cash in Wallets with Observers
Advances in Cryptology – CRYPTO '93
Springer Verlag, 1994, pp. 302-318.

[31] Bressoud, David M.
Factorization and Primality Testing
UTM, Springer, 1989.

[32] Brown, Julian
The Quest of the Quantum Computer
Simon & Schuster
New York, London 2000.

[33] Buchmann, Johannes
Einführung in die Kryptographie
2., erweiterte Auflage
Springer Verlag
Berlin, Heidelberg, New York, 2001.

[34] Buchmann, Johannes
Faktorisierung großer Zahlen
Spektrum der Wissenschaft
September 9/1996, Seite 80 - 88.

[35] Budiansky, Stephen
Battle of Wits
The Complete Story of Codebreaking in World War II
Simon & Schuster
New York, 2002.

[36] Charlap, Leonard S. and David P. Robbins
An Elementary Introduction to Elliptic Curves I
CRD Expository Report 31
Center for Communications Resarch, Princeton, 1988.

[37] Charlap, Leonard S. and Raymond Coley
An Elementary Introduction to Elliptic Curves II
CRD Expository Report 34
Center for Communications Resarch, Princeton, 1990.

[38] Chaum, David
Untraceable electronic mail, return addresses, and digital pseudonyms
Communications of the ACM **24** (1981) 84 – 88.

[39] Chaum, David
Blind signatures for untraceable payments
Advances in Cryptology – CRYPTO '82
D. Chaum, R.L. Rivest, A.T. Sherman (Eds.)
Plenum Press, New York, London, 1983, p. 199–203.

[40] Chaum, David
The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability
J. Cryptology (1988) **1**, 65 – 75.

[41] Chaum, David
Security without Identification: Transaction Systems to Make Big Brother Obsolete
Communications of the ACM **28** (1985) 10, 1030 - 1044
www.chaum.com/articles/Security-without-Identification.html

[42] Chaum, David
Achieving Electronic Privacy
Scientific American, August 1992, p. 96 – 101.

[43] Chaum, David, Amos Fiat and Moni Naor
Untraceable Electronic Cash
in: S. Goldwasser (Hrsg.) Advances in Cryptology – CRYPTO '88
Springer Verlag, (1988), 319 – 327.

[44] Chaum, David, Eugene van Heijst and Birgit Pfitzmann
Cryptographically strong undeniable signatures, unconditionally secure for the signer
Advances in Cryptology – CRYPTO '91
LNCS 576, Springer Verlag, 1992, pp. 470-484.

[45] Churchhouse, Robert
Codes and Ciphers
Julius Caesar, the Enigma, and the internet
Cambridge University Press, Cambridge, 2002.

[46] Cohen, Henri
Zahlentheoretische Aspekte der Kryptographie
Informatik Spektrum
Juni 2001, Seite 129 - 139.

[47] Comer, Douglas E.
Internetworking with TCP/IP
Volume 1
Principles, Protocols, and Architectures
Fourth Edition
Prentice Hall
Upper Saddle River, New Jersey, 2000.

[48] Copeland, B. Jack
The Essential Turing
The ideas that gave birth to the computer age
Clarendon Press
Oxford, 2004.

[49] Copeland, B. Jack *et al.*
Colossus
The Secrets of Bletchley Park's Codebreaking Computers
Oxford University Press
Oxford 2006.

[50] Copeland, B. Jack
Turing
Pioneer of the Information Age
Oxford University Press, New York, 2012.

[51] Cormen, Thomas, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
Introduction to Algorithms
Second Edition
The MIT Press
Cambridge, Massachusetts, 2001.

[52] Crandall, Richard and Carl Pomerance
Prime Numbers
A Computational Perspective
Springer Verlag
Berlin, Heidelberg, New York, 2001.

[53] Daemen, Joan and Vincent Rijmen
AES Proposal: Rijndael
FIPS PUB 80.

[54] Daemen, Joan and Vincent Rijmen
The Design of Rijndael
AES – The Advanced Encryption Standard
Springer Verlag
Berlin, Heidelberg, 2002.

[55] Dankmeier, Wilfried
Codierung
(Fast) alles über Daten-Verschlüsselung, Kompression und Fehlerbeseitigung
2. Auflage
Friedrich Vieweg Verlag
Braunschweig, Wiesbaden, 2001.

[56] *Data Encryption Standard*
Federal Information Processing Standard (FIPS)
Publication 46
National Bureau of Standards,
U.S. Department of Commerce

Washington D.C., January 1977.
http://www.itl.nist.gov/fipspubs/fip46-2.htm

[57] DE SAUTOY, MARCUS
Die Musik der Primzahlen
Auf den Spuren des größten Rätsels der Mathematik
C. H. Beck
München, 2004.

[58] *DES Modes of Operation*
Federal Information Processing Standard (FIPS)
Publication 81
National Bureau of Standards,
U.S. Department of Commerce
Washington D.C., December 1980.
http://www.itl.nist.gov/fipspubs/fip81.htm

[59] DEVLIN, KEITH
Sternstunden der modernen Mathematik
Berühmte Probleme und neue Lösungen
Birkhäuser Verlag
Basel, Boston, Berlin, 1990.

[60] DEWDNEY, A.K.
Der Algorithmus des Data Encryption Standard
Spektrum der Wissenschaft
Januar 1/1989, Seite 6.

[61] DEWDNEY, A.K.
Der Turing Omnibus
Eine Reise durch die Informatik in 66 Stationen
Springer Verlag
Berlin, Heidelberg, New York, 1995.

[62] DIETZFELBINGER, MARTIN
Primality Testing in Polynomial Time
From Randomized Algorithms to 'PRIMES is in P'
LNCS 3000, Springer, 2004.

[63] DIFFIE, WHITFIELD
The First Ten Years of Public-Key Cryptography
Proceedings of the IEEE
Vol. 76, **5** (1988), 560 - 577.

[64] DIFFIE, WHITFIELD and MARTIN E. HELLMAN
New Directions in Cryptography
IEEE Transactions on Information Theory
IT-22, **6** (1976), 644-654.
Siehe auch:
http://www.cs.purdue.edu/homes/ninghui/courses//Fall04/lectures/diffie-hellman.pdf

[65] DIFFIE, WHITFIELD and MARTIN E. HELLMAN
Exhaustive Cryptanalysis of the NBS Data Encryption Standard
Computer, June 1977, pp. 74 – 84.

[66] Diffie, Whitfield and Martin E. Hellman
Privacy and Authentication: An Introduction to Cryptography
Proceedings of the IEEE
Vol. 67, **3** (1979), pp. 397 - 427.

[67] Diffie, Whitfield and Susan Landau
Privacy on the Line
The Politics of Wiretapping and Encryption
The MIT Press
Cambridge, MA, London, 1998.

[68] Dittmann, Jana
Digitale Wasserzeichen
Grundlagen, Verfahren, Anwendungsgebiete
Springer Verlag
Berlin, Heidelberg, New York, 2000.

[69] Dittmann, Jana, Elke Franz und Antje Schneidewind
Steganographie und Wasserzeichen
Aktueller Stand und neue Herausforderungen
Informatik Spektrum
Dezember 2005, Seite 453 - 461.

[70] Donovan, Peter ans John Mack
Code Breaking in the Pacific
Springer Verlag, NY, 2014.

[71] Dunham, William
Journey through Genius
The Great Theorems of mathematics
Pengiun Books
New York, 1990.

[72] Dworkin, Morris
Recommendation for Block Cipher Modes of Operation
Methods and Techniques
NIST Special Publication 800-38A (2001)

[73] Eastlake, D., S. Crocker, J. Schiller
Randomness Recommendations for Security
RFC 1750, December 1994.

[74] Eckert, Claudia
IT–Sicherheit
Konzepte – Verfahren – Protokolle
3. Auflage
Oldenbourg Verlag
München, Wien, 2004.

[75] ElGamal, Taher
A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms
IEEE Transactions on Information Theory
IT-31, **4** (1985), 469-472.

[76] ElGamal, Taher
A Subexponential Time Algorithm for Computing Discrete Logarithms over $GF(p^2)$
IEEE Transactions on Information Theory
IT-31, **4** (1985), 472-481.

[77] Erskine, Ralph and Michael Smith (ed.)
The Bletchley Park Codebreakers
Biteback Publishing, London, 2011.

[78] Ertel, Wolfgang
Angewandte Kryptographie
Fachbuchverlag Leipzig im Carl Hanser Verlag
München, Wien, 2001.

[79] Fahrmeir, Ludwig, Rita Künstler, Iris Pigeot und Gerhard Tutz
Statistik
Der Weg zur Datenanalyse
Dritte Auflage
Springer Verlag
Berlin, Heidelberg, New York, 2001.

[80] Feistel, Horst
Cryptography and Computer Privacy
Scientific American
May 1973, pp.15 - 23 .

[81] Ferguson, Niels, and Bruce Schneier
Practical Cryptography
John Wiley & Sons. 2003.

[82] Fiat, Amos and Adi Shamir
How to prove yourself: Practical Solutions to Identification and Signature Problems
Advances in Cryptology - Crypto '86
LNCS 263, Springer Verlag.

[83] Fischer, Volker Gerd und Volker Grossmann
Abgehorcht
Public–Key–Infrastrukturen im Gesundheitswesen
iX – Magazin für professionelle Informationstechnik
September 2001, pp. 52 – 55.

[84] Fluhrer, S., I. Martin and Adi Shamir
Weaknesses in the Key Scheduling Algorithm of RC4
Eighth Annual Workshop on Selected Areas in Cryprography
August 2001.

[85] Forster, Otto
Algorithmische Zahlentheorie
Friedrich Vieweg Verlag
Braunschweig/Wiesbaden, 1996.

[86] W.F. Friedman
The Index of Coincidence and its Applications in Cryptography
Dept. of Ciphers Publ. 22, Illinois, 1922.

[87] Gaines, Helen Fouchee
Cryptanalysis
A Study of Ciphers and Their Solution
Dover Publications
New York, 1956.

[88] Galbraith, Steven and Alfred Menezes
Algebraic Curves and Cryptography
Preprint, erscheint in: *Finite Fields and Their Applications*

[89] Gannon, Paul
Colossus – Bletchley Park's Greatest Secret
Atlantic Books, London, 2006.

[90] Gardner, Martin
Mathematical Games
A new kind of cipher that would take millions of years to break
Scientific American, Vol. 237, **8**, Aug. 1977, pp. 120–124.

[91] Garey, Michael R. and David S. Johnson
Computers and Intractability
A Guide to the Theory of $\mathcal{NP}$-Completeness
W. H. Freeman and Company
New York, 1979.

[92] Garfinkel, Simson and Gene Spafford
Web Security, Privacy & Commerce
Second Edition
O'Reilly
Beijing, Cambridge, 2002.

[93] Garfinkel, Simson, Gene Spafford and Alan Schwartz
Practical Unix & Internet Security
Third Edition
O'Reilly
Beijing, Cambridge, 2003.

[94] Garrett, Paul
Making, Breaking Codes: An Introduction to Cryptology
Prentice Hall
Upper Saddle River, New Jersey, 2001.

[95] Gioia, Anthony A.
The Theory of Numbers
An Introduction
Dover Publications, Inc.
New York, 2001.

[96] Goldreich, Oded
Foundations of Cryptography — A Primer
Foundations and Trends in Theoretical Computer Sciene, Vol. 1, No. 1, 1 - 116 (2005).

[97] Goldreich, Oded
Foundations of Cryptographyy
Basic Tools
Cambridge University Press
Cambridge, 2001.

[98] Goldwasser, Shafi, Silvio Micali and Charles Rackoff
The Knowledge Complexity of Interactive Proof-Systems
Prc. of 17th Annual ACM Symposium on Theory of Computing (STOC), SIAM 1989.

[99] Gollman, Dieter and William G. Chambers
Clock–Controlled Shift Registers: A Review
IEEE Journal on Selected Areas in Communications, Vol. 7, No. 4, (1989), 525 – 533.

[100] Golomb, Solomon W.
Shift Register Sequences
Holden–Day, Inc.
San Francisco, Cambridge, London, Amsterdam, 1967.

[101] Gruska, Josef
Quantum Computing
McGraw-Hill
London, 1999.

[102] Hamer, David H.
G–312: An Abwehr Enigma
Cryptologia **24**, 1 (January 2000)

[103] Harel, David
Das Affenpuzzle
und weitere bad news aus der Computerwelt
Springer Verlag
Berlin, Heidelberg, 2002.

[104] Harel, David und Yishai Feldman
Algorithmik
Die Kunst des Rechnens
Springer Verlag
Berlin, Heidelberg, 2006.

[105] Hayes, Brian
The Magic Words are Squeamish Ossifrage
American Scientist **82**, 1994, 312 – 316.

[106] Hellman, Martin E.
Die Mathematik neuer Verschlüsselungssysteme

Spektrum der Wissenschaft
Oktober 10/1979, Seite 93 – 101.

[107] Hankerson, Darrel, Alfred Menezes and Scott Vanstone
Guide to Elliptic Curve Cryptography
Springer Verlag
New York, 2004.

[108] Lester S. Hill
Cryptography in an algebraic alphabet
Amer. Math. Monthly **36** (1929), 306–311.

[109] Lester S. Hill
Concerning certain linear transformation apparatus of cryptography
Amer. Math. Monthly **38** (1931), 135–154.

[110] Hinsley F.H. and Alan Stripp (Ed.)
Codebreakers
The inside story of Bletchley Park
Oxford University Press
Oxford, 1993.

[111] Hodges, Andrew
Alan Turing, Enigma
Zweite Auflage
Springer-Verlag
Wien, New York, 1994.

[112] Hoffstein, Jeffrey, Jill Pipher, and Joseph H. Silverman
An Introduction to Mathematical Cryptography
Undergraduate Texts in Mathematics
Springer–Verlag
New York, 2008.

[113] Holden, Joshua
The Mathematics of Secrets
Cryptography from Caesar Ciphers to Digital Encryption
Princeton University Press, Princeton NJ, 2017.

[114] Hornfeck, Bernhard
Algebra
Walter de Gruyter
Berlin, 1969.

[115] Ianoco, Luigi Lo and Niko Schweitzer
Kaffeesatz
Java-Crypto-Provider im Vergleich
iX – Magazin für professionelle Informationstechnik
Oktober 2001, pp. 74 – 78.

[116] Janowicz, Krzysztof
Sicherheit im Internet
O'Reilly Verlag
Köln, 2002.

[117] Jurisic, Aleksandar and Alfred J. Menezes
Elliptic Curves and Cryptography
Dr. Dobb's Journal, April 1997, 26 – 35.

[118] Kahn, David
The Codebreakers
The Comprehensive History of Secret Communication from Ancient Times to the Internet
Scribner, New York, 1996.

[119] Kahn, David
Hitler's Spies
German Military Intelligence in World War II
Da Capo Press, 2000.

[120] Kahn, David
Seizing the Enigma
Arrow Books
London, 1992.

[121] Kaliski Jr., Burton S., Ronald L. Rivest, and Alan T. Sherman
Is the Data Encryption Standard a Group? (Results of Cycling Experiments on DES)
J. Cryptology (1988) **1** 3 – 36.

[122] Kent, S. and R. Atkinson
IP Encapsulating Security Payload (ESP)
RFC 2406, November 1998.

[123] Kippenhahn, Rudolph
Verschlüsselte Botschaften
Geheimschrift, Enigma und Chipkarte
Rowohlt Verlag
Reinbek bei Hamburg, 1997.

[124] Klein, D.
Foiling the Cracker: A Survey of, and Improvements to Password Security
Proceedings UNIX Security Workshop II
August 1990.

[125] Knuth, Donald E.
The Art of Computer Programming Vol. 1
Fundamental Algorithms
Third Edition
Addison Wesley
Reading, Massachusetts, 1997.

[126] Knuth, Donald E.
The Art of Computer Programming Vol. 2
Seminumerical Algorithms
Third Edition
Addison Wesley
Reading, Massachusetts, 1998.

[127] Knuth, Donald E.
The Art of Computer Programming Vol. 3
Sorting and Searching
Second Edition
Addison Wesley
Reading, Massachusetts, 1998.

[128] Koblitz, Neal
Elliptic curve cryptosystems
Mathematics of Computation, **48** (1987), 203 – 209.

[129] Koblitz, Neal
A Course in Number Theory and Cryptography
Second Edition
Graduate Texts in Mathematics, 114
Springer Verlag
New York, 1994.

[130] Koblitz, Neal
Algebraic Aspects of Cryptography
Algorithms and Computation in Mathematics
Volume 3
Springer Verlag
New York, 1997.

[131] Koblitz, Neal
The Uneasy Relationship Between Mathematics and Cryptography
Notices of the AMS, Vol. 54, No. 8, 972 – 979 (2007).

[132] Koblitz, Neal and Alfred J. Menezes
A Survey of Public-Key Cryptosystems
SIAM Review, **46** (2004), 599 – 634.

[133] Koy Henrik und Jörg Schneider
Selbst geknackt
Spielerisches Erforschen der Kryptopgraphie
c't – Magazin für Computertechnik
Heft 14, 2001, Seite 204 – 209.

[134] Krieger, Wolfgang
Geschichte der Geheimdienste
Von den Pharaonen bis zur CIA
Verlag C.H. Beck
München, 2010.

[135] Kruh, Louis and Cipher Deavours
The Commercial Enigma: Beginnings of Machine Cryptography
Cryptologia **25**, 1 (January 2002).

[136] **Kryptographie**
Spektrum der Wissenschaft
Dossier
Verlag Spektrum der Wissenschaft
Heft 4/2001.

[137] Lang, Serge
Undergraduate Algebra
Third Edition
Springer Verlag
New York, 2005.

[138] Leavitt, David
The Man Who Knew Too Much
Alan Turing and the Invention of the Computer
W.W. Norton Company
New York, London, 2006.

[139] Leiberich, Otto
Vom diplomatischen Code zur Falltürfunktion
Spektrum der Wissenschaft
Juni 6/1999, Seite 26 – 34.

[140] Lenstra, A.K., H.W. Lenstra, Jr., M.S. Manasse and J. M. Pollard
The Factorization of the Ninth Fermat Number
Mathematics of Computation **61**, 203, (1993), 319 – 349.

[141] Lenstra, Hendrik W.
Factoring Integers with elliptic curves
Annals of Mathematics, **126** (1987), 649 – 673.

[142] LeVeque, William J.
Elementary Theory of Numbers
Dover Publications, Inc.
New York, 1990.

[143] Levy, Steven
Crypto
How the Code Rebels Beat the Government
Saving Privacy in the Digital Age
Viking Penguin
New York, London, 2001.

[144] Lidl, Rudolf and Harald Niederreiter
Introduction to finite fields and their applications
Revised Edition
Cambridge University Press
Cambridge, 1994.

[145] Lipmaa, Helger, Phillip Rogaway and David Wagner
Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption
2000

[146] Lovász, László, József Pelikán und Katalin Vesztergombi
Diskrete Mathematik
Springer Verlag
Berlin, Heidelberg, New York, 2005.

[147] Luckhardt, Norbert
Pretty Good Privacy
Teil 1: Einstieg in das Web of Trust
c't – Magazin für Computertechnik
Heft 12, 1999, Seite 212 – 214.

[148] Luckhardt, Norbert
Pretty Good Privacy
Teil 2: Schlüsselfragen und -antworten
c't – Magazin für Computertechnik
Heft 13, 1999, Seite 208 – 210.

[149] Luckhardt, Norbert
Pretty Good Privacy
Teil 3: Dateibearbeitung und geteilte Schlüssel
c't – Magazin für Computertechnik
Heft 16, 1999, Seite 172 – 175.

[150] Maurer, Ueli and James L. Massey
Cascade Ciphers: The Importance of Being First
Journal of Cryptology, Vol. 6, No. 1, pp. 55 – 61 (1993).

[151] McClain, Sally
Navajo Weapon
The Navajo Code Talkers
Rio Nuevo Publisher, Tucson Texas, 1994.

[152] Meijer, Alko R.
Algebra for Crytologists
Springer, 2016.

[153] Menezes, Alfred J.
Elliptic Curve Public Key Cryptosystems
Kluwer Academic Publishers
Boston, Dordrecht, London, 1993.

[154] Menezes, Alfred J., Paul van Oorschot, Scott A. Vanstone
Handbook of Applied Cryptography
CRC Press, 1996.

[155] Merkle, Ralph C. and Martin E. Hellman
Hiding Information and Signatures in Trapdoor Knapsacks
IEEE Transactions on Information Theory
IT-24, **5** (1978), 525-530.

[156] Mermin, N. David
Quantum Computer Science
An Introduction
Cambridge University Press
Cambridge, UK, 2007.

[157] Miller, Victor S.
Use of elliptic curves in cryptography

Advances in Cryptology – CRYPTO '85
LNCS 218, (1986) 417 – 426.

[158] Mishra, Arunseh and William A. Arbaugh
An Initial Security Analysis of the IEEE 802.1X Standard
University of Maryland
Tech. Report CS-TR-4328
Feb 6, 2002
URL: http://www.cs.umd.edu/~waa/papers.shtml

[159] Musa, Mohammed, Edward F. Schaefer and Stephen Wedig
A Simplified Rijndael Algorithm and Its Linear and Differential Cryptanalysis
Cryptologia, **27**, 2, 148 – 177, (2003).

[160] Nielsen, Michael A. and Isaac L. Chuang
Quantum Computation and Quantum Information
Cambridge University Press
Cambridge, 2000.

[161] Ochel, David und Oliver Weissmann
Schriftgut
PKI soll sichere Kommunikation gewährleisten
iX – Magazin für professionelle Informationstechnik
September 2001, pp. 44 – 50.

[162] Odlyzko, Andrew M.
The future of integer factorisation
CryptoBytes **1** (no. 2) (1995) pp. 5 – 12.

[163] Okamoto, Tatsuaki and Kazuo Ohta
Universal electronic cash
Advances in Cryptology – CRYPTO '91,
Lecture Notes in Computer Science 576,
Springer Verlag, 1992, pp. 324 - 337.

[164] Christof Paar, Jan Pelzl
Understanding Cryptography
A Textbook for Students and Practionioners
Springer, Berlin, Heidelberg, 2010.

[165] Papadimitriou, Christos H.
Computational Complexity
Addison Wesley Longman, Reading, Massachusetts, 1994.

[166] Peterson, Ivars
Mathematische Expeditionen
Ein Streifzug durch die Mathematik
Spektrum Akademischer Verlag
Heidelberg, 1998.

[167] Charles Petzold
The Annotated Turing

A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine
Wiley Publishing
Indianapolis, Indiana, 2008.

[168] Phan, Raphael Chung–Wei
Mini Advanced Encryption Standard (Mini–AES): A Testbed for Cryptanalysis Students
Cryptolgia, 26:4, 283 – 306 (2002).

[169] Pincock, Stephen, Mark Frary
Geheime Codes
Die berühmtesten Verschlüsselungstechniken und ihre Geschichte
Verlagsgruppe Lübbe
Bergisch Gladbach, 2007

[170] Pöppe, Christoph
Späte Rehabilitation des Data Encryption Standard
Spektrum der Wissenschaft
Mai 5/1993, Seite 19 – 24.

[171] Pollard, John M.
A Monte Carlo Method For Factorization
Nordisk Tidskrift för Informationsbehandlung (BIT) **15** (1975), 331 – 334.

[172] Pomerance, Carl
Primzahlen im Schnelltest
Spektrum der Wissenschaft
Februar 2/1983, Seite 80 - 92.

[173] Quisquater, J.-J. and L.C. Guillou
How to explain Zero-Knowledge to Your Children
Advances in Cryptology – CRYPTO '89 Proceedings
Springer Verlag, 1990, pp. 629 – 631.

[174] Ratcliff, R.A.
Delusions of Intelligence
Enigma, Ultra, and the End of Secure Ciphers
Cambridge University Press
Cambridge, 2006.

[175] Reiss, K. und G. Schmieder
Basiswissen Zahlentheorie
Eine Einführung in Zahlen und Zahlenbereiche
Springer Verlag
Berlin, Heidelberg, New York, 2005.

[176] Rembold, Ulrich und Paul Levi
Einführung in die Informatik
3. Auflage
Carl Hanser Verlag
München, Wien, 1999.

[177] Remmert, Reinhold und Peter Ullrich
Elementare Zahlentheorie
Dritte Auflage
Birkhäuser
Basel, Boston, Berlin, 2008.

[178] Renyi, Alfred
Wahrscheinlichkeitsrechnung
mit einem Anhang über Informationstheorie
Fünfte Auflage
Deutscher Verlag der Wissenschaften
Berlin, 1977.

[179] Ribenboim, Paulo
The Book of Prime Number Records
Springer-Verlag
New York, Berlin, Heidelberg, 1988.

[180] Richter, R., P. Sander und W. Stucky
Der Rechner als System
Organisation, Daten, Programme
B.G. Teubner, Stuttgart 1997.

[181] Rivest, Ronald L.
Cryptography
in:
Handbook of Theoretical Computer Science, Vol. A
J. van Leeuwen, Ed.
Elsevier Science Publisher B.V.,
Amsterdam, 1990.

[182] Rivest, Ronald L., Adi Shamir and Len Adleman
A Method for Obtaining Digital Signatures and Public Key Cryptosystems
Communications of the ACM
Vol. 21, (1978), pp. 120 – 126.
Siehe auch:
http://theory.lcs.mit.edu/~rivest/rsapaper.pdf

[183] Robshaw, M.J.B.
Stream Ciphers
RSA Labs Technical Report TR–701, July 1995.

[184] Ronce, C.A.
Feedback Shift Registers
LNCS 169
Springer Verlag
Berlin, 1984.

[185] Rothe, Jörg
Some Facets of Complexity Theory and Cryptography: A Five-Lecture Tutorial
ACM Computing Surveys, **34**, No. 4 (2002), 504 – 549.

[186] ROTHMAN, TONY
Das kurze Leben des Evariste Galois
Spektrum der Wissenschaft
Juni 6/1982, Seite 102 - 112.

[187] RSA Laboratories
RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1
RSA Security Inc., 2000.

[188] Rueppel, Rainer A.
Stream Ciphers
in: G.J. Simmons, editor. *Contemporary Cryptology, The Science of Information Integrity*, pp. 65 – 134.
IEEE Press
New York, 1992.

[189] SCHICKINGER, THOMAS und ANGELIKA STEGER
Diskrete Strukturen 2
Wahrscheinlichkeitstheorie und Statistik
Springer Verlag
Berlin, Heidelberg, 2001.

[190] SCHMEH, KLAUS
Kryptografie
und Public-Key-Infrastrukturen im Internet
2., aktualisierte und erweiterte Auflage
dpunkt.verlag
Heidelberg, 2001.

[191] SCHMEH, KLAUS
Die Welt der geheimen Zeichen
Die faszinierende Geschichte der Verschlüsselung
W3L Verlag Herdecke
Dortmund, 2004.

[192] SCHMEH, KLAUS
Versteckte Botschaften
Die faszinierende Geschichte der Steganographie
Heise, Hannover, 2009.

[193] SCHNEIER, BRUCE
Applied Cryptography
Protocols, Algorithms and Source Code in C
John Wiley & Sons, Inc.
New York, 1996.

[194] SCHNEIER, BRUCE
Secrets & Lies
Digital Security in a Networked World
John Wiley & Sons, Inc.
New York, 2000.

[195] Schroeder, M.R.
Number Theory in Science and Communication
With Applications in Cryptography, Physics, Digital Information, computing and Self-Similarity
Third Edition
Springer Verlag
Berlin, Heidelberg, New York, 1999.

[196] Schulzki–Haddouti, Christiane und Andreas Weisse
Elektriktrick
Chiffriermaschinen des 20. Jahrhunderts
c't – Magazin für Computertechnik
Heft 3, 2000, Seite 110 – 119.

[197] Sebag–Montefiore, Hugh
Enigma
The Battle For The Code
Phoenix, London, 2000.

[198] Shannon, Claude E.
A Mathematical Theory of Communication
Bell Systems Technical Journal
Vol. **27**, 379 – 423, 623 – 656, 1948.

[199] Shannon, Claude E.
Communication Theory of Secrecy Systems
Bell Systems Technical Journal
No. **4**, 1949.

[200] Shannon, Claude E. and Warren Weaver
The Mathematical Theory of Communication
University of Illinois Press
Urbana, Chicago, 1949, 1998.

[201] Shor, Peter W.
Algorithms for quantum computation: discrete logarithms and factoring
In: *Proceedings of the* 35^{th} *Annual Symposum on Foundation of Computer Science*
IEEE Press, Los Alamitos, CA, 1994.

[202] Silverman, Joseph H.
The Arithmetic of Elliptic Curves
Graduate Texts in Mathematics
Springer-Verlag
New York, 1986.

[203] Silverman, Joseph H. and John Tate
Rational Points on Elliptic Curves
Undergraduate Texts in Mathematics
Springer-Verlag
New York, 1992.

[204] Singh, Simon
Geheime Botschaften
Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet
Carl Hanser Verlag
München, Wien, 2000.

[205] Stallings, William
Cryptography and Network Security
Second Edition
Prentice Hall
Upper Saddle River, New Jersey, 1999.

[206] Stallings, William
Cryptography and Network Security
Fourth Edition
Prentice Hall
Upper Saddle River, New Jersey, 2006.

[207] Stallings, William
Cryptography and Network Security
Sixth Edition
Prentice Hall
Upper Saddle River, New Jersey, 2014.

[208] Stallings, William
Cryptography and Network Security
Principles and Practice
Seventh Edition
Prentice Hall
Pearson, 2017.

[209] Stallings, William
The Advanced Enryption Standard
Cryptologia, 26:3, 165–188 (2002).

[210] Stallings, William and Lawrie Brown
Computer Security
Principles and Practice
Pearson, Prentice Hall
Upper Saddle River, New Jersey, 2008.

[211] Stephens, N.N.
Lenstra's Factorisation Method Based on Elliptic Curves
Advances in Cryptology – CRYPTO '85
LNCS 218, (1986) 409 – 416.

[212] Stillwell, John
Elements of Number Theory
UTM
Springer, New York, 2003.

[213] Stinson, Douglas R.
Cryptography; Theory and Practice

Second Edition
Chapman & Hall/CRC
Boca Raton, London, 2002.

[214] Stubblefield, Adam, John Ioannidis and Aviel D. Rubin
Using the Fluhrer, Mantin and Shamir Attack to Break WEP
ATT Labs Technical Report TD 42CPZZ
August 2004.

[215] Trappe, Wade and Lawrence C. Washington
Introduction to Cryptography with Coding Theory
Prentice Hall
Upper Saddle River, New Jersey, 2002.

[216] van der Lubbe, Jan C.A.
Basic Methods of Cryptography
Cambridge University Press, Cambridge, 1997.

[217] van der Waerden, B.L.
Algebra I
Vierte Auflage
Springer Verlag
Berlin, Göttingen, Heidelberg, 1955.

[218] Wätjen, Dietmar
Kryptographie
Grundlagen, Algorithmen, Protokolle
Spektrum Akademischer Verlag
Heidelberg, Berlin, 2003.

[219] Waldecker, Rebecca, Rempe–Gillen, Lasse
Primzahltests für Einsteiger
2. Auflage
Springer–Verlag, 2016.

[220] Walker, Jesse
Unsafe at any key size; An analysis of the WEP encapsulation
Tech. Rep 03628E, IEEE 802.11 committee, March 2000
URL:http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/0-362.zip

[221] Warner, Seth
Modern Algebra
Dover Publications, Inc.
New York, 1990.

[222] Washington, Lawrence C.
Elliptic Curves
Number Theory and Cryptography
Chapmann & Hall/CRC
Boca Raton, Florida, 2003.

[223] Martin H. Weissman
An Illustrated Theory of Numbers
American Mathematical Society, 2017.

[224] Welchman, Gordon
The Hut Six Story
Breaking the Enigma Codes
M& M Baldwin
Cleobury Mortimer, Shropshire, 2005.

[225] Welschenbach, Michael
Cryptography in C and C++
Second Edition
Apress/Springer-Verlag
New York, 2005.

[226] Wendt, Patrick
Wenn lang nicht langt
Jahrhundert-Hack; RSA-129 geknackt
iX – Magazin für professionelle Informationstechnik
September 1994, pp. 130 – 137.

[227] Werner, Annette
Elliptische Kurven in der Kryptographie
Springer Verlag
Berlin, Heidelberg, New York, 2002.

[228] Wiener, M.
Efficient DES Key Search
Proceedings of the Crypto '93
Springer Verlag
Heidelberg, New York, 1993.

[229] Williams, Collin P., Scott Clearwater
Ultimate Zero and One
Computing at the Quantum Frontier
Copernicus, Springer-Verlag
New York, 2000.

[230] Winkel, Brian J., David Kahn, Cipher Deavors, Louis Kruh
The German Enigma Machine
Beginnings, Success, and Ultimate Failure
Artech House
Boston, London, 2005.

[231] Winterbotham, F.W.
The Ultra Secret
The Inside Story of Operation Ultra, Bletchley Park and Enigma
Orion Paperback
London, 2001.

[232] Witt, Kurt–Ulrich
Algebraische Grundlagen der Informatik
Zahlen, Strukturen, Codierung, Verschlüsselung
Friedrich Vieweg Verlag
Braunschweig, Wiesbaden, 2001.

[233] Wobst, Reinhard
Von DES zu AES
Neuer Verschlüsselungsstandard vor der Verabschiedung
iX – Magazin für professionelle Informationstechnik
Oktober 2001, pp. 92 – 96.

[234] Wobst, Reinhard
AES unter Beschuss
Code-Breaker feiern 'Durchbruch'
c't – Magazin für Computertechnik
Heft 21, 2002, Seite 38.

[235] Wrixon, Fred B.
Geheimsprachen
Codes, Chiffren und Kryptosysteme von den Hieroglyphen zum Digitalzeitalter
Könemann, Tandem Verlag, 2006.

[236] Yardley, Herbert O.
The American Black Chamber
American House, 1931.

[237] Zeng, K., C.-H. Yang, D.-Y. Wei, T.R.N. Rao
Pseudorandom Bit Generators in Stream-Cipher Cryptography
Computer, February 1991, p. 8 – 16.

[238] Zimmermann, Phillip R.
Schlüsseldienst fürs Internet
Spektrum der Wissenschaft
März 3/1999, Seite 95 – 99.

Online Quellen

[239] RSA Laboratories
http://www.rsasecurity.com/rsalabs/faq/index.html

[240] RFC
http://www.rfc-editor.org

[241] Counterpane (Bruce Schneier)
http://www.counterpane.com

[242] VPN Consortium
http://www.vpnc.org

[243] Enigma
http://www.codesandciphers.org.uk

[244] Bletchley Park
http://www.bletchleypark.org.uk

[245] Ron Rivest Security Page and his home page
Ron Rivest's security page contains a huge amount of links.
http://theory.lcs.mit.edu/~rivest/crypto-security.html
http://theory.lcs.mit.edu/~rivest/

[246] Phil Zimmermanns home page
http://web.mit.edu/prz

[247] NSA Kryptologie Museum
http://www.nsa.gov/museum/index.html

[248] CERT Coordination Center
http://www.cert.org

[249] The Prime Pages
http://www.utm.edu/research/primes

[250] Die NIST AES Homepage
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdfi

[251] Der Data Encryption Standard
http://www.itl.nist.gov/fipspubs/fip46-2.htm

[252] Cryptographic Module Validation
http://csrc.ncsl.nist.gov/cryptval

[253] Bundesamt für Sicherheit in der Informationstechnik
http://www.bsi.de

[254] Cryptool: Freie Software zur Demonstration verschiedener Verschlüsselungsalgorithmen
http://www.cryptool.de

[255] Electronic Frontier Foundation
http://www.eff.org/Privacy

[256] Gesellschaft für Informatik, Fachbereich Sicherheit – Schutz und Zuverlässigkeit
http://www.gi-ev/fachbereiche/sicherheit

[257] David Kahns Homepage
http://david-kahn.com

[258] Whitfield Diffies Homepage
http://research.sun.com/people/diffie

[259] Martin Hellmans Homepage
http://www-ee.stanford.edu/~hellman/

[260] IBMs Website über MARS
http://domino.research.ibm.com/comm/research_projects.nsf/pages/security.mars.html

[261] Website von Certicom
http://www.certicom.com

Index

3DES, 100

Abel, Niels Henrik, 412
Abelsche Gruppe, 412
Abgeschlossenheit, 353, 410, 445
Adams, Carlisle, 101
Addition modulo n, 443
    Definition, 444
ADFGVX–Chiffre, 60
ADFGVX-Chiffre, 64
ADFGX–Chiffre, 60
Adleman, Leo, 223, 553
Advanced Encryption Standard, 101, 114, 185–203, 651
Äquivalenzrelation, 439
Äquivalenzklasse, 442, 476
Äquivalenzrelation, 498
    Reflexivität, 498
    Symmetrie, 498
    Transitivität, 499
AES, 20, *siehe* Advanced Encryption Standard, 185, 692
    Inverse S–Box, 194
    S–Box, 193
    Zustand, 189
Affine Chiffre, 40–46, 70, 416
    Definition, 40
Agrawal, Manindra, 589
Algebraische Struktur, 410
Algebraisches System, 410
Algorithmus, 626
    Euklidischer, 532
    Diffie-Hellman Key Exchange, 235
    Euklidischer für Polynome, 665
    Baby-Step-Giant-Step von Shanks, 616
    BBS Generator, 578
    Berechnung der Quadratwurzel modulo n, 517
    Elliptic Curve Diffie Hellman, 394
    Erweiterer Euklidischer, 379
    Erweiterter Euklidischer, 57, 222, 227, 232, 241, 537, 632, 655
    Erweiterter Euklidischer für Polynome, 669, 715
    Erweiterter Euklidischer, 523
    Euklidischer, 433, 523
    Faktorisierung, 551
    J. Stein, 546
    Korrektheit, 465
    kryptographischer, 10
    Modulare Exponentiation, 238, 460
    Rijndael, 186, 715
    RSA, 225
    Stein, 546–549
    subexponentialer, 639
    Teilermethode, 556, 638
    Terminierung, 465
Alphabet, 411
AND Operation, 447
Anderson, Ross, 101
Anderson, Ross, 187
ANSI, 29, 114
ANSI X9.55, 29
Arithmetik modulo n, 450
ASC X9, 29
Assoziativgesetz, 150, 354, 410–412, 420, 423, 494
Attacke
    Brute Force, 15, 112, 147, 167, 232, 331
    Chosen–ciphertext, 15, 17
    Chosen–plaintext, 15, 16
    Ciphertext–only, 15
    Geburtstags-, 288
    Known–plaintext, 15, 16, 44, 81
    man-in-the-middle, 237

Message Stream Modifikation, 144
Timing, 233
Aubexponentialzeit Algorithmus, 639
Authentifikation, 7, 147, 212, 287
Authentifizierung, 25, 206
Authentikator, 22, 213, 292
Authentizität, 6
Automorphismus, 484
Autorisierung, 7
Avalanche Effekt, 132–133, 763

Babbage, Charles, 47
BB84, 399
BBS generator, 578
Benioff, Paul, 399
Bennett, Charles H., 398
Bigramm, 99
Biham, Eli, 101
Biham, Eli, 187
Bijektion, 480
Binäre Addition, 627
Binäre Division, 628
Binäre Multiplikation, 627
Binäre Subtraktion, 627
Binäre Suche, 642
Bit Commitment, 207
Bit-Commitment Protokolle, 277–278
Bit-Operation, 624
Bit-Operationen, 624–626
Bletchley Park, 23
Blinde Signatur, 265, 280
eingeschränkte, 271
Blinde Signaturen, 268, 272–275
Block Chiffre, 20, 58, 97, 101, 103
Block Chiffre, iterative, 197
Block replay, 137
Block-Wiederholung, 137
Blowfish, 31, 100, 103, 162–168
Dechiffrierung, 166
S-Box, 163, 166
Verschlüsselungsalgorithmus, 165
Blum-Blum-Shub Pseudorandom Bit Generator, 578
Bombe, 94
Brands, Stefan, 264
Brassard, Gilles, 399
Brute-Force Attacke, 219, 420, 572
BSI, 31
Bundesdatenschutzgesetz, 7

Cäsar Chiffre, 37, 414
Camellia, 103
Carmichael Zahlen, 590
CAST–128, 101, 103
CAST–256, 101
Cayley Tafel, 477
Cayley, Arthur, 482
CBC, *siehe* Modus, 183, 184
Certicom, 334
Challenge and Response, 237
Charakteristisches Polynom, 751
Chaum, David, 32
Chaum, David, 264
Chiffre, 10
monoalphabetische, 39, 43, 47, 54
polyalphabetische, 49
Chiffretext, 10
Chiffrierung, 10
Chinesischer Restesatz, 226, 467–471, 517
Chinesischer Restsatz, 467
Church-Turing These, 397
Cocks, Clifford, 23
Cocks, Clifford, 206
Codebook, 135
Codebuch, 68
Cohen, Henri, 334
Coin Flip, 207
Common Criteria, 32
Computer Security Resource Center, 30
Coppersmith, Don, 103
coprim, 241, 434, 457, 523
crypt(3), 146
Cryptogramm, 10
Cryptotext, 10
Cyclic Redundancy Code, 176
Cyperpunks, 169

Daemen, Joan, 101
Daemen, Joan, 186
Data Encryption Standard, 103, 111, 303, 572
Datenintegrität, 175
Datenschutz, 7
Datensicherheit, 7
Dechiffrierung, 10
DES, 20, 27, *siehe* Data Encryption Standard, 185, 187, 572

DEUTSCH, DAVID, 399
DIFFIE, WHITFIELD, 23
Diffie, Whitfield, 113, 206, 215, 223
Diffie–Hellman Key Exchange, 27
Diffie-Hellman Key Exchange, 234–383
Diffie-Hellman Schlüsselaustausch, 234
Diffusion, 59, 99, 133, 150, 152
Digital Cash, 263–284
Digital Signature Standard, 316, 334
Digitale Signatur, 6, 22, 23, 206, 213, 214, 240, 265, 267, 287, 294, 296, 301
Digitale Wasserzeichen, 9
Digitale Zeitstempel, 289
Dining Cryptographers Problem, 285
Diophantische Gleichung, 335
Diophantus, 336
Diskreter Logarithmus, 234, 240, 262, 615–619
  Definition, 234
Diskretes Logarithmusproblem, 236, 244, 245, 254, 262, 283, 346, 383
  elliptische Kurven, 356
Distributivgesetz, 150, 423
Division modulo n, 456
Divisionsalgorithmus, 661
Divisionsring, 422
DSA, 572
DSS, 316, 572

ECB, 183
Echelon, 30
EFF, 31
Einheiten einer Menge, 479
Einheitengruppe, 479
Einwegfunktion, 206
Electronic Cash, 24
Electronic code book, 183
Electronic Frontier Foundation, 114
Elektonische Münze, 265
Elektronische Wahlen, 24
ELGAMAL Verfahren, 27
ElGamal Kryptosystem, 240–254, 262, 383
ElGamal Kryptosysteme, 346
ElGamal, Taher, 240
Elliptische Kurve
  Definition, 343
  singuläre, 344
Elliptische Kurve über endlicher Körpern, 357
Elliptische Kurven, 3, 551
  Faktorisierung, 378–382
  Kryptosysteme, 383–395
Elliptische Kurven Kryptographie
  Diffie-Hellman Key Exchange, 394
  ElGamal Kryptosystem, 385
ELLIS, JAMES, 23
Ellis, James, 206
Endliche Körper, 3, 424
Endlicher Körper, 446
  Ordnung, 653
Endomorphismus, 484
Enigma, 83
  Codebuch, 89
  Spruchschlüssel, 89
Entscheidungsproblem, 641
  polynomiale Laufzeit, 640
Entschlüsselung, 10
Epimorphismus, 484
Erzeugende Funktion, 717
Euklid, 429, 585
EULER, LEONHARD, 427, 434, 552
EULERsche Totientenfunktion, 43, 434
Euler Konstante, 180
Euler Theorem, 224
Eulersche Totientenfunktion, 224, 499, 612
Eulersches Kriterium, 368, 513
Exponentialzeit Algorithmus, 638

Faktorisierung, 26, 383, 431
Fast Data Encipherment Algorithm, 103
FEAL, *siehe* Fast Data Encipherment Algorithm
Feedback FunKtion, 717
Feedback Shift Register, 726
Fehlererkennung, 300
FEISTEL Chiffre, 101
Feistel Chiffre, 103–110, 127, 162, 167
Feistel Netzwerk, 103
Feistel structure, 182
Feistel, Horst, 104, 111
FERMAT, PIERRE DE, 552
FERMAT Zahlen, 452

Fermat number, 157
Fermat Zahl, 638
Fermat Zahlen, 552
Fermat, Pierre de, 501
Fermatsches Theorem, 226, 248, 502, 512
FEYNMAN, RICHARD, 398
Fiat, Amos, 258
Fiat–Shamir Protokoll, 508
Fiat-Shamir Protokoll, 258
FIPS, 187
FIPS PUB 180, 326
FIPS PUB 180-1, 326
FIPS Pub 186-2, 334
FIPS PUB 46, 111
Fundamentalsatz der Zahlentheorie, 430

Galois Feld
  Addition, 693
  Multiplikation, 697
  Polynome, 697
Galois Felder, 651
Galois field, 201, 473
Galois, Evariste, 473, 652
GaloisGALOIS, EVARISTE, 651
Ganze Zahlen modulo n, 444
Gauss Klammer, 584
Gauss, Carl F., 585
GCHQ, 23, 206
Geburtstags Paradoxon, 311–315, 563
Geburtstagsangriff, 306, 310–315
Geheimhaltung, 6
Geheimtinte, 9
Gemeinsamer Teiler, 431
Gemeinsames Vielfaches, 431
Goldener Schnitt, 180
Goldwasser, Shafi, 255
Größter gemeinsamer Teiler, 418, 431
GROVER, LOV, 399
Gruppe, 412
  abelsche, 491, 494, 695
  allgemeine lineare, 742
  Definition, 475
  Erzeugendes Element, 490
  multiplikative, 246, 248, 601
  Ordnung, 476
  Potenzschreibweise, 490
  unendlich zyklische, 491
  zyklische, 490, 499
Gruppe,
  zyklische, 477
Gruppe, abelsche, 354, 485, 490
Gruppenhomomorphismen, 484–489
Gruppenhomorphismus
  Definition, 484

Häufigkeitsverteilung, 54
Halbgruppe, 410
Hash Funktion, 175, 206, 241, 281, 287, 292, 300
  Anforderungen, 305
  kollisionsfreie, 288
  One-Way, 300
  Sicherheit, 288
Hash Wert
  Definition, 305
Hash-Wert, 292, 300
HDLC, 298
HELLMAN, MARTIN E., 23
Hellman, Martin E., 113, 206, 215, 223
HILL, LESTER S., 56
Hill Chiffre
  Definition, 59
Histiaeus, 9
HMAC, 317
Homomorphismus, 484
Hybridverfahren, 27, 206

IDEA, 20, 100
  Multiplikation/Additions Struktur, 152
Idempotenz, 487
Identification friend or foe, 17
Identifikation, 7, 147
Identifikationssysteme, 24
IEEE, 30
IEEE 1363, 30
IEEE P1363, 334
IFF, 17
Impuls Response Sequenz, 744
Index, 234, 383, 615
Induktion, 465
Informationstheorie, 71
Injektivität, 43
Instanz eines Problems, 640
Integrität, 6, 213, 297, 299
Integritätsbereich, 422

Interaktiver Zero-Knowledge Beweis, 255
International Data Encryption Algorithm, 149
International Organization for Standardization, 30
Inverses
  additives, 423
  multiplikatives, 41, 57, 423, 446, 523, 534, 535, 543, 651
Inverses Element, 412, 423, 475
ISO, 29, *siehe* International Organization for Standardization
ISO/OSI Modell, 299
Isomorphismus, 484
ITSEC, 32

Körper, 423, 446, 651
  endlicher, 240, 651
Kartesisches Produkt, 410
KASISKI Test, 47
KasiskiKASISKI, FRIEDRICH W., 47
KAYAL, NEERAJ, 589
Kerckhoffs Prinzip, 18, 211, 305
Kerckhoffs, Auguste, 18
Kern, 489
Key exchange, 206
Key Exchange Problem, 22
Key Management Problem, 22
Klartext, 8
Klein, Felix, 489
Kleines FERMATsches Theorem, 589
Kleinsche Vierergruppe, 489
Kleinstes gemeinsames Vielfaches, 431
Knapsack System, 27
KNUDSEN, LARS, 101
Knudsen, Lars, 187
Koblitz, Neal, 333, 383
Koeffizient
  führender, 659
Kollision, 288
Kollisionsfreiheit, 315
Kollisionsresistenz
  schwache, 305
  starke, 305
Kommunikationsmodell, 5
Kommutativgesetz, 165, 353, 412, 423, 445
Komplexitätsklasse
  $\mathcal{NP}$, 220
  $\mathcal{P}$, 216
Komplexitätsklasse $co - \mathcal{NP}$, 645
Komplexitätsklasse $\mathcal{NP}$, 645
Komplexitätsklasse $\mathcal{NP}$-vollständig, 648
Komplexitätsklasse $\mathcal{P}$, 589, 644
Komplexitätstheorie, 216
Kompressions Funktion, 320
Konfusion, 99, 133, 150
Konkatenation, 411
Korrektheit des RSA Algorithmus, 225
Kosten, 13
Kryptoanalyse, 4, 55, 90, 233
Kryptographie, 4, 415
Kryptologe, 4
Kryptologie
  Definition, 4
Kryptosystem
  Definition, 12
Kryptosystem, endomorphisches, 68

Lai, X., 149
Lai, Xuejia, 100
LANDAUER, ROLF, 398
Laufzeitbestimmung
  Algorithmen, 626–632
Left Shift, 118
Legendre, 585
Lehmer, D.H., 573, 588
Leiberich, Otto, 32
Lemma von BÉZOUT, 433
Lenstra, Hendrik, 333
LFSR, 716
Linear Feedback Shift Register, 716–723
Lineare Rückkopplungsrelation, 726
Lineare Rückkopplungssequenzen
  Initial Vektor, 729
  Zustandsvektor, 729
Linksnebenklasse, 497
Logarithmus, 486
Longitudinal Redundancy Check, 307
Lucifer, 103, 111

MAC, *siehe* Message Authentication Code, 292, 297
Magenta, 103
Man in the middle Attacke, 5
MARS, 103, 187

Maskierung, 290
Massey, J., 149
Massey, James L., 100
Mauborgne, Joseph O., 79
MD4, 316, 766
MD5, 175, 316, 330
Merkle, Ralph C., 206
MERSENNE, MARTIN, 431
Mersenne Primzahlen, 431
Message Authentication Code, 21, 292, 297
Message Digest, 213, 288, 300, 317
Message encryption, 292
MEYER, CARL, 111
Micali, Silvio, 255
Mikrophotographie, 9
Miller-Rabin Test, 231, 591–600
MISTY1, 103
Modification
  sequence, 290
  timing, 290
Modukare Arithmetik, 443
Modulare Arithmetik, 44, 229, 414, 450
Modulare Exponentation, 458
Modus
  cipher block chaining, 140, 310
  cipher feedback, 142
  counter, 145
  electronic codebook, 135
  Output Feedback, 144
  output feedback, 142
Monoid, 411
Monte Carlo Methode von POLLARD, 551
MOORE, GORDON, 398
Mooresches Gesetz, 398
Multiplikation modulo n, 443
  Definition, 444
Multiplikationschiffre, 69
Multiplikatives Inverses, 42

Nachrichtenmodifikation, 290
Navajo–Code, 8
Nebenklassen
  Definition, 496
NESSIE, 31
Neutrales Element, 412, 423, 475
  Addition, 445
  Multiplikation, 446
NIST, 30, 101, 111, 113, 134, 145, 185, 309, 316, 326, 334
NSA, 29, 111, 113, 114, 316
Nullteiler, 479

OFB, *siehe* Modus
Offenlegung, 290
One–Time Pad, 14, 71–82, 99
  Definition, 80
One–Way Funktion, 206
One-Time Pad, 276, 572, 719
One-Way Funktion, 277, 305
One-Way Hash Funktion, 278, 305
ongruenzen, 439
Operation
  binäre, 410, 443, 651
Operation, binäre, 346
Orakel, 647
Ordnung eines Elements, 601

Pad, 81
Padding, 184
Perfekte Sicherheit, 75
Permutation, 65, 115, 480
Permutation, inverse, 65
Permutationen, 12
Permutations Chiffre, 65
  Definition, 65
Permutationschiffre, 65
Permutationsgruppe, 482
PGP, 149, 576
PKI, 237
Playfair Chiffre, 51
POLLARD, JOHN M., 563
POLYBIOS, 60
Polybios–Quadrat, 60
Polynom, 651
  ggT, 665
  Euklidischer Algorithmus, 665–668
  Grad, 659
  irreduzible, 697
  irreduzibles, 677, 678, 710
  kanonische Faktorisierung, 679
  Koeffizient, 656
  monic, 659
  normiertes, 659
  reduzibles, 678
  relativ prim, 665
  Teiler, 664

Wurzel, 687
Polynome, 421
Produkt, 657
Summe, 657
Polynome über einem Körper, 659
Polynomialzeit Algorithmus, 637
Polynomring, 659
Prüfsumme, kryptographische, 297
Pretty Good Privacy, 100
Prim–Polynom, 677
Primites Element, 735
Primitive Wurzel, 234, 246, 248, 281
Definition, 610
Primzahl, 428
Primzahl Test, 587–600
Primzahlen, 3, 583
Primzahlsatz, 232, 585
Primzahltest, 501, 563
Lehmann, 589
Miller-Rabin, 589
Solovay-Strassen, 589
Miller-Rabin, 591–600
Private Key, 23, 209, 223, 241
Produkt Kryptosystem, 68
Protokolle, kryptographische, 207
Pseudo-Primzahlen, 589
Pseudozufallszahl, 573
Pseudozufallszahlen, 572–582
Pseudozufallszahlengenerator, 82, 97
kryptographisch sicherer, 575
Public Key, 23, 209, 223
Public key, 241
Public Key Kryptosystem, 333
Public–Key Kryptographie, 23, 429
Public–Key Kryptosystem, 17, 208
Public-Key Infrastruktur, 237
Public-Key Kryptosystem, 288, 583
Definition, 208

Quadratischer Nichtrest, 258, 508
Quadratischer Rest, 258, 358, 368, 508–514
Quadratwurzel modulo n, 508, 514–521
Quantenkanal, 404
Quantenkryptographie, 397–406
Quantenteleportation, 399
Quantum key distribution, 399
Qubit, 404
Quisquater, Jean-Jacques, 256

Rückkopplungsfunktion, 716, 717
Rackoff, Charles, 255
RC2, 101
RC4, 20, 101, 169–175
RC5, 101, 103, 178–184
block cipher mode, 183
CBC mode, 183
CBC-Pad mode, 183
CTS mode, 184
RC6, 187
Rechtsnebenklasse, 497
Reduktion modulo f, 684
Rejewski, Marian, 91
Relation, 439
Relativ prim, 434
Repudiation, 290
Restklassen, 442
Restklassen modulo n, 442
RFC 1321, 317, 324
RFC 3174, 326
Rijmen, Vincent, 101
Rijmen, Vincent, 186
Rijndael, 101
ByteSub transormation, 201
Key expansion, 201
Key schedule, 202
MixColumn operation, 202
ShiftRow Operation, 202
State, 198
Ring, 421, 659
Assoziativitätsgesetz, 421
Distributivgesetz, 421
Kommutativgesetz, 422
Ring mit Identität, 422
RIPEMD, 316
RIPEMD-160, 330
Rivest, Ron, 6, 20, 101, 553
Rivest, Ron, 3, 31, 103, 169, 178, 187, 223, 316
Rotation, 118, 178, 308
Rotor Maschinen, 83–94
RSA, 20, 572, 583
RSA Algorithmus, 223–229, 505, 553
RSA Challenge, 233, 553–555
RSA Public–Key Kryptosystem, 27
RSA Security, 31, 101
RSA Verfahren, 223, 244
Rundenfunktion, 105
Rundenschlüssel, 104

S–Box, 123
S-Box, 113, 128, 150, 163, 169, 201
Satz von CAYLEY, 482
Satz von EUKLID, 429
Satz von EULER, 504
Satz von LAGRANGE, 498
Satz von Bayes, 72, 77
SAXENA, NITIN, 589
Scherbius, Arthur, 83
Schieberegister, 716
    erzeugende Funktion, 717
    Periode, 716
Schiefkörper, 422
Schlüssel, 10, 415
Schlüsselaustausch, 206
Schlüsselaustausch Problem, 22, 205
Schlüsselmanagement, 21
Schlüsselmanagement Problem, 22
Schlüsselraum, 10, 42, 48, 112
Schlüsselstrom, 97
Schlüsselwort, 47
Schleifeninvariante, 465
SCHNEIER, BRUCE, 100
Schneier, Bruce, 31, 103, 162, 187, 316
(m, n)-Schwellenwertproblem, 276
Secret sharing, 207
Secret Splitting, 276
Secure Hash Algorithm, 316, 326–330
Senderanonymität, 285
Sequenz
    linear rückgekoppelte, 725
    periodische, 733
    ultimativ periodische, 732
    Vorperiode, 733
Serpent, 101, 187
Session Key, 27, 215
SHA, 316, 326
SHA-1, 326
Shamir, Adi, 31, 223, 258, 553
SHANNON, CLAUDE E., 12, 78, 99
SHANNON. CLAUDE E., 14
Shannon, Claude E., 68, 71, 80, 106, 415
Shift Chiffre, 37–39, 69, 76, 414
    Definition, 37
Shift Operation, 116
Shift Register, 142
SHOR, PETER W., 399
Sicherheit
    beweisbare, 14
    uneingeschränkte, 14
Simple Network Management Protocol, 299
Skytale, 36
Splitting field, 752
Steganographie, 8
STEIN, JOSEF, 546
Strom Chiffre, 20, 97, 101, 142, 145, 169, 716, 717
Subset Problem, 218
Substitution, 12, 60, 83, 100, 115, 123
Substitutions Chiffre, 39
Substitutionschiffre, 54, 65
Suchproblem, 641
Sukzessives Verdoppeln, 355
Superencipherment, 19
Superencryption, 19
    Kaskadierung, 19
    Mehrfache Verschlüsselung, 19

Tartaglia, Niccolo, 256
TAVARES, STAFFORD, 101
TEA, 103
Teiler, 425
Teilermethode, 551
Teilnehmerauthentifikation, 255
Traffic Analysis, 290
Transposition, 12
Transpositionschiffre, 36
Trap–door Einwegfunktion, 216
Trapdoor Einwegfunktion, 26
Traveling Salesman Problem, 640
Traveling Salesman Problem, exaktes, 649
Trigramm, 99
Triple DES, 100, 576
Triple–DES, 185
TUCHMAN, WALTER, 111
TURING, ALAN, 83
Turing Maschine, 397
Turing, Alan, 397
Twofish, 31, 103, 187

Ultra, 84
Untergruppe, 489
    Definition, 494
    triviale, 494

Untergruppen, 494–496

Vandermonde Determinante, 752
Verbindlichkeit, 6, 23, 206
Verkehrsflussanalyse, 290
Vernam, Gilbert, 79
Verschlüsselung, 10
    Asymmetrische, 20
    Private Key, 20
    Secret key, 20
    Symmetrische, 11, 20, 296
Vertraulichkeit, 212, 292
Vielfaches, 425
Vigenére Chiffre, 47–49
Vigenère Chiffre
    Definition, 47
Vigenère Tabelle, 50
Vigenère, Blaise de, 47
Virtual Private Network, 768
Vollkommene Zahlen, 426
von Neumann Rechnerarchitektur, 398
von Neumann, John, 398

Wahrscheinlichkeit
    a posteriori, 72, 76
    a priori, 71, 76
    bedingte, 72
Weierstrass Gleichung, 344
WEP, *siehe* Wired Equivalent Privacy
WEP Verschlüsselung, 101
Wheatstone, Sir Charles, 51
Williamson, Malcolm, 23
Williamson, Malcolm, 206
Winterbotham, Frederick W., 84
Wired Equivalent Privacy, 20, 175
Word, 178
Wort, 411

XOR, 120
XOR Operation, 109, 447
XTEA, 103

Zahl
    Länge, 621
    zusammengestzte, 428
Zahlen
    ganze, 410
    natürliche, 409
    rationale, 413
Zahlenfeldsieb, 551
Zahlenfeldsieb, quadratisches, 551
Zahlentheorie, 410
Zeilinger, Anton, 399
Zerfällungskörper, 751, 752
Zero knowledge proof, 207
Zero-Knowledge Beweis, 255
Zero-Knowledge Protokoll
    Simulator, 261
Zero-Knowledge Protokolle, 255–262
Zertifikat, 237, 289
Zimmermann, Phil, 32
Zufallsvariable, 71
Zufallszahlengenerator, 572
Zusammengesetzte Zahl, 430