

$2^2 - 1 = 3$  prim

$2^3 - 1 = 7$  prim

$2^5 - 1 = 31$  prim

$2^7 - 1 = 127$  prim

$2^{11} - 1 = 2047 = 23 \cdot 89$

# Das CrypTool-Skript

## Kryptographie, Mathematik und mehr

Prof. Bernhard Esslinger  
und das CrypTool Entwickler-Team

10. Auflage

Hintergrundmaterial und Zusatzinformationen  
zum freien E-Learning-Programm CrypTool  
(mit Code-Beispielen zur Zahlentheorie, geschrieben in Sage)

# Das CrypTool-Skript: Kryptographie, Mathematik und mehr

Hintergrundmaterial und Zusatzinformationen  
zum freien E-Learning-Programm CrypTool  
(mit Code-Beispielen zur Zahlentheorie, geschrieben in Sage)

(10. Auflage – veröffentlicht mit CrypTool-Version 1.4.30)

(c) Prof. Bernhard Esslinger (Mitautor und Herausgeber)  
und das CrypTool Entwickler-Team, 1998-2010  
Frankfurt am Main

[www.cryptool.org](http://www.cryptool.org)

5. Juli 2010

Dies ist ein freies Dokument, d.h. Inhalte des Dokuments können kopiert und verbreitet werden, auch zu kommerziellen Zwecken.

Im Gegenzug sind Autor, Titel und die CrypTool-Webseite ([www.cryptool.org](http://www.cryptool.org)) zu nennen. Selbstverständlich darf aus dem CrypTool-Skript, genau wie aus jedem anderen Werk auch, zitiert werden.

Dieses Dokument unterliegt der GNU-Lizenz für freie Dokumentation.

Copyright © 1998–2010 Bernhard Esslinger and the CrypTool Development Team. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Quelle Coverfoto: [www.photocase.com](http://www.photocase.com), Andre Günther

Schriftsatz-Software: L<sup>A</sup>T<sub>E</sub>X

Versionsverwaltungs-Software: Subversion

# Überblick über den Inhalt des CrypTool-Skripts

In diesem *Skript zu dem Programm CrypTool* finden Sie eher mathematisch orientierte Informationen zum Einsatz von kryptographischen Verfahren. Zu einigen Verfahren gibt es Beispielcode, geschrieben für das Computer-Algebra-System **Sage** (siehe Anhang A.5). Die Hauptkapitel sind von verschiedenen **Autoren** verfasst (siehe Anhang A.2) und in sich abgeschlossen. Am Ende der meisten Kapitel finden Sie jeweils Literaturangaben und Web-Links.

Das erste Kapitel beschreibt die Prinzipien der symmetrischen und asymmetrischen **Verschlüsselung** und erläutert kurz die aktuellen Entschlüsselungsrekorde bei modernen symmetrischen Verfahren.

Im zweiten Kapitel wird – aus didaktischen Gründen – eine ausführliche Übersicht über **Papier- und Bleistiftverfahren** gegeben.

Ein großer Teil des Skripts ist dem faszinierenden Thema der **Primzahlen** (Kapitel 3) gewidmet. Anhand vieler Beispiele bis hin zum **RSA-Verfahren** wird in die **modulare Arithmetik** und die **elementare Zahlentheorie** (Kapitel 4) eingeführt.

Danach erhalten Sie Einblicke in die mathematischen Konzepte und Ideen hinter der **modernen Kryptographie** (Kapitel 5).

Kapitel 6 gibt einen Überblick zum Stand der Attacken gegen moderne **Hashalgorithmen** und widmet sich dann kurz den **digitalen Signaturen** — sie sind unverzichtbarer Bestandteil von E-Business-Anwendungen.

Kapitel 7 stellt **Elliptische Kurven** vor: Sie sind eine Alternative zu RSA und für die Implementierung auf Chipkarten besonders gut geeignet.

Das letzte Kapitel **Krypto 2020** diskutiert Bedrohungen für bestehende kryptographische Verfahren und stellt alternative Forschungsansätze für eine langfristige kryptografische Sicherheit vor.

Während das *eLearning-Programm CrypTool* eher den praktischen Umgang motiviert und vermittelt, dient das *Skript* dazu, dem an Kryptographie Interessierten ein tieferes Verständnis für die implementierten mathematischen Algorithmen zu vermitteln – und das didaktisch möglichst gut nachvollziehbar. Wer sich schon etwas mit der Materie auskennt, kann mit dem **Menübaum** (siehe Anhang A.1) einen schnellen Überblick über die Funktionen in CrypTool gewinnen.

Die Autoren möchten sich an dieser Stelle bedanken bei den Kollegen in der Firma und an den Universitäten Frankfurt, Gießen, Siegen, Karlsruhe und Darmstadt.

Wie auch bei dem E-Learning-Programm CrypTool wächst die Qualität des Skripts mit den Anregungen und Verbesserungsvorschlägen von Ihnen als Leser. Wir freuen uns über Ihre Rückmeldung.

# Kurzinhaltsverzeichnis

Überblick	ii
Vorwort zur 10. Auflage des CrypTool-Skripts	xi
Einführung – Zusammenspiel von Skript und CrypTool	xii
1 Verschlüsselungsverfahren	1
2 Papier- und Bleistift-Verschlüsselungsverfahren	14
3 Primzahlen	52
4 Einführung in die elementare Zahlentheorie mit Beispielen	95
5 Die mathematischen Ideen hinter der modernen Kryptographie	178
6 Hashfunktionen und Digitale Signaturen	191
7 Elliptische Kurven	200
8 Krypto 2020 — Perspektiven für langfristige kryptographische Sicherheit	221
A Anhang	226
GNU Free Documentation License	256
Abbildungsverzeichnis	264
Tabellenverzeichnis	265
Verzeichnis der Krypto-Verfahren	267
Verzeichnis der Sage-Programmbeispiele	268
Index	270

# Inhaltsverzeichnis

Überblick	ii
Vorwort zur 10. Auflage des CrypTool-Skripts	xi
Einführung – Zusammenspiel von Skript und CrypTool	xii
<b>1 Verschlüsselungsverfahren</b>	<b>1</b>
1.1 Symmetrische Verschlüsselung . . . . .	2
1.1.1 Neue Ergebnisse zur Kryptoanalyse von AES . . . . .	2
1.1.2 Aktueller Stand der Brute-Force-Angriffe auf symmetrische Verfahren (RC5) . . . . .	4
1.2 Asymmetrische Verschlüsselung . . . . .	5
1.3 Hybridverfahren . . . . .	6
1.4 Chiffren und Kryptoanalyse für Lehrzwecke . . . . .	7
1.5 Weitere Details . . . . .	7
1.6 Anhang: Beispiele mit Sage . . . . .	8
1.6.1 Mini-AES . . . . .	8
Literaturverzeichnis . . . . .	10
Web-Links . . . . .	13
<b>2 Papier- und Bleistift-Verschlüsselungsverfahren</b>	<b>14</b>
2.1 Transpositionsverfahren . . . . .	15
2.1.1 Einführungs-Beispiele unterschiedlicher Transpositionsverfahren . . . . .	15
2.1.2 Spalten- und Zeilentranspositionsverfahren . . . . .	16
2.1.3 Weitere Transpositionsverfahren . . . . .	18
2.2 Substitutionsverfahren . . . . .	20
2.2.1 Monoalphabetische Substitutionsverfahren . . . . .	20
2.2.2 Homophone Substitutionsverfahren . . . . .	23
2.2.3 Polygraphische Substitutionsverfahren . . . . .	24
2.2.4 Polyalphabetische Substitutionsverfahren . . . . .	26
2.3 Kombination aus Substitution und Transposition . . . . .	28
2.4 Andere Verfahren . . . . .	32

2.5	Anhang: Beispiele mit Sage . . . . .	34
2.5.1	Transpositions-Chiffren . . . . .	35
2.5.2	Substitutions-Chiffren . . . . .	39
2.5.3	Caesar-Chiffre . . . . .	40
2.5.4	Verschiebe-Chiffre . . . . .	42
2.5.5	Affine Chiffren . . . . .	43
2.5.6	Substitutions-Chiffre mit Symbolen . . . . .	45
2.5.7	Vigenère-Verschlüsselung . . . . .	47
2.5.8	Hill-Verschlüsselung . . . . .	48
	Literaturverzeichnis . . . . .	50
<b>3</b>	<b>Primzahlen</b>	<b>52</b>
3.1	Was sind Primzahlen? . . . . .	52
3.2	Primzahlen in der Mathematik . . . . .	53
3.3	Wie viele Primzahlen gibt es? . . . . .	54
3.4	Die Suche nach sehr großen Primzahlen . . . . .	55
3.4.1	Die 20+ größten bekannten Primzahlen (Stand Juli 2009) . . . . .	55
3.4.2	Spezielle Zahlentypen – Mersennezahlen und Mersenne-Primzahlen . . . . .	57
3.4.3	Wettbewerb der Electronic Frontier Foundation (EFF) . . . . .	60
3.5	Primzahltests . . . . .	60
3.6	Übersicht Spezial-Zahlentypen und die Suche nach einer Formel für Primzahlen . . . . .	63
3.6.1	Mersennezahlen $f(n) = 2^n - 1$ für $n$ prim . . . . .	63
3.6.2	Verallgemeinerte Mersennezahlen $f(k, n) = k \cdot 2^n \pm 1$ / Proth-Zahlen . . . . .	63
3.6.3	Verallgemeinerte Mersennezahlen $f(b, n) = b^n \pm 1$ / Cunningham-Projekt . . . . .	63
3.6.4	Fermatzahlen $f(n) = 2^{2^n} + 1$ . . . . .	64
3.6.5	Verallgemeinerte Fermatzahlen $f(b, n) = b^{2^n} + 1$ . . . . .	65
3.6.6	Carmichaelzahlen . . . . .	65
3.6.7	Pseudoprimzahlen . . . . .	65
3.6.8	Starke Pseudoprimzahlen . . . . .	65
3.6.9	Idee aufgrund von Euklids Beweis $p_1 \cdot p_2 \cdots p_n + 1$ . . . . .	65
3.6.10	Wie zuvor, nur $-1$ statt $+1$ : $p_1 \cdot p_2 \cdots p_n - 1$ . . . . .	66
3.6.11	Euklidzahlen $e_n = e_0 \cdot e_1 \cdots e_{n-1} + 1$ . . . . .	66
3.6.12	$f(n) = n^2 + n + 41$ . . . . .	67
3.6.13	$f(n) = n^2 - 79 \cdot n + 1.601$ . . . . .	67
3.6.14	Polynomfunktionen $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$ . . . . .	68
3.6.15	Vermutung von Catalan . . . . .	69
3.7	Dichte und Verteilung der Primzahlen . . . . .	69
3.8	Anmerkungen zu Primzahlen . . . . .	72

3.8.1	Bewiesene Aussagen / Sätze zu Primzahlen . . . . .	72
3.8.2	Unbewiesene Aussagen / Vermutungen zu Primzahlen . . . . .	75
3.8.3	Offene Fragen zu Primzahlzwillingen . . . . .	77
3.8.4	Weitere offene Fragestellungen . . . . .	78
3.8.5	Kurioses und Interessantes zu Primzahlen . . . . .	78
3.9	Anhang: Anzahl von Primzahlen in verschiedenen Intervallen . . . . .	80
3.10	Anhang: Indizierung von Primzahlen ( $n$ -te Primzahl) . . . . .	81
3.11	Anhang: Größenordnungen / Dimensionen in der Realität . . . . .	82
3.12	Anhang: Spezielle Werte des Zweier- und Zehnersystems . . . . .	83
3.13	Anhang: Visualisierung der Menge der Primzahlen in hohen Bereichen . . . . .	84
3.14	Anhang: Beispiele mit Sage . . . . .	88
3.14.1	Einfache Funktionen zu Primzahlen mit Sage . . . . .	88
3.14.2	Primalitäts-Check der von einer quadratischen Funktion erzeugten Zahlen . . . . .	89
	Literaturverzeichnis . . . . .	91
	Web-Links . . . . .	93
	Dank . . . . .	94
<b>4</b>	<b>Einführung in die elementare Zahlentheorie mit Beispielen</b>	<b>95</b>
4.1	Mathematik und Kryptographie . . . . .	95
4.2	Einführung in die Zahlentheorie . . . . .	96
4.2.1	Konvention . . . . .	98
4.3	Primzahlen und der erste Hauptsatz der elementaren Zahlentheorie . . . . .	99
4.4	Teilbarkeit, Modulus und Restklassen . . . . .	101
4.4.1	Die Modulo-Operation – Rechnen mit Kongruenzen . . . . .	101
4.5	Rechnen in endlichen Mengen . . . . .	103
4.5.1	Gesetze beim modularen Rechnen . . . . .	103
4.5.2	Muster und Strukturen . . . . .	104
4.6	Beispiele für modulares Rechnen . . . . .	105
4.6.1	Addition und Multiplikation . . . . .	105
4.6.2	Additive und multiplikative Inversen . . . . .	106
4.6.3	Potenzieren . . . . .	109
4.6.4	Schnelles Berechnen hoher Potenzen . . . . .	110
4.6.5	Wurzeln und Logarithmen . . . . .	111
4.7	Gruppen und modulare Arithmetik über $\mathbb{Z}_n$ und $\mathbb{Z}_n^*$ . . . . .	112
4.7.1	Addition in einer Gruppe . . . . .	112
4.7.2	Multiplikation in einer Gruppe . . . . .	113
4.8	Euler-Funktion, kleiner Satz von Fermat und Satz von Euler-Fermat . . . . .	115
4.8.1	Muster und Strukturen . . . . .	115



4.8.2	Die Euler-Funktion . . . . .	115
4.8.3	Der Satz von Euler-Fermat . . . . .	116
4.8.4	Bestimmung der multiplikativen Inversen . . . . .	116
4.8.5	Fixpunkte modulo 26 . . . . .	117
4.9	Multiplikative Ordnung und Primitivwurzel . . . . .	118
4.10	Beweis des RSA-Verfahrens mit Euler-Fermat . . . . .	121
4.10.1	Grundidee der Public-Key-Kryptographie . . . . .	121
4.10.2	Funktionsweise des RSA-Verfahrens . . . . .	122
4.10.3	Beweis der Forderung 1 (Umkehrbarkeit) . . . . .	123
4.11	Zur Sicherheit des RSA-Verfahrens . . . . .	125
4.11.1	Komplexität . . . . .	125
4.11.2	Sicherheitsparameter aufgrund neuer Algorithmen . . . . .	126
4.11.3	Vorhersagen zur Faktorisierung großer Zahlen . . . . .	127
4.11.4	Status der Faktorisierung von konkreten großen Zahlen . . . . .	129
4.11.5	Weitere aktuelle Forschungsergebnisse zu Primzahlen und Faktorisierung . . . . .	134
4.12	Anwendungen asymmetrischer Kryptographie mit Zahlenbeispielen . . . . .	138
4.12.1	Einwegfunktionen . . . . .	138
4.12.2	Das Diffie-Hellman Schlüsselaustausch-Protokoll (Key Exchange Protocol) . . . . .	139
4.13	Das RSA-Verfahren mit konkreten Zahlen . . . . .	142
4.13.1	RSA mit kleinen Primzahlen und mit einer Zahl als Nachricht . . . . .	142
4.13.2	RSA mit etwas größeren Primzahlen und einem Text aus Großbuchstaben . . . . .	142
4.13.3	RSA mit noch etwas größeren Primzahlen und ASCII-Zeichen . . . . .	143
4.13.4	Eine kleine RSA-Cipher-Challenge (1) . . . . .	146
4.13.5	Eine kleine RSA-Cipher-Challenge (2) . . . . .	146
4.14	Anhang: Der ggT und die beiden Algorithmen von Euklid . . . . .	149
4.15	Anhang: Abschlussbildung . . . . .	151
4.16	Anhang: Bemerkungen zur modulo Subtraktion . . . . .	151
4.17	Anhang: Basisdarstellung von Zahlen, Abschätzung der Ziffernlänge . . . . .	152
4.18	Anhang: Beispiele mit Sage . . . . .	155
4.18.1	Multiplikationstabellen modulo $m$ . . . . .	155
4.18.2	Schnelles Berechnen hoher Potenzen . . . . .	155
4.18.3	Multiplikative Ordnung . . . . .	156
4.18.4	Primitivwurzeln . . . . .	159
4.18.5	RSA-Beispiele mit Sage . . . . .	169
4.18.6	Wie viele RSA-Schlüssel gibt es innerhalb eines Modulo-Bereiches? . . . . .	170
4.19	Anhang: Liste der in diesem Kapitel formulierten Definitionen und Sätze . . . . .	172
	Literaturverzeichnis . . . . .	173
	Web-Links . . . . .	176

Dank . . . . .	177
<b>5 Die mathematischen Ideen hinter der modernen Kryptographie</b>	<b>178</b>
5.1 Einwegfunktionen mit Falltür und Komplexitätsklassen . . . . .	178
5.2 Knapsackproblem als Basis für Public-Key-Verfahren . . . . .	180
5.2.1 Knapsackproblem . . . . .	180
5.2.2 Merkle-Hellman Knapsack-Verschlüsselung . . . . .	181
5.3 Primfaktorzerlegung als Basis für Public-Key-Verfahren . . . . .	182
5.3.1 Das RSA-Verfahren . . . . .	182
5.3.2 Rabin-Public-Key-Verfahren (1979) . . . . .	184
5.4 Der diskrete Logarithmus als Basis für Public-Key-Verfahren . . . . .	185
5.4.1 Der diskrete Logarithmus in $\mathbb{Z}_p^*$ . . . . .	185
5.4.2 Diffie-Hellman-Schlüsselvereinbarung . . . . .	186
5.4.3 ElGamal-Public-Key-Verschlüsselungsverfahren in $\mathbb{Z}_p^*$ . . . . .	187
5.4.4 Verallgemeinertes ElGamal-Public-Key-Verschlüsselungsverfahren . . . . .	187
Literaturverzeichnis . . . . .	190
<b>6 Hashfunktionen und Digitale Signaturen</b>	<b>191</b>
6.1 Hashfunktionen . . . . .	192
6.1.1 Anforderungen an Hashfunktionen . . . . .	192
6.1.2 Aktuelle Angriffe gegen Hashfunktionen wie SHA-1 . . . . .	193
6.1.3 Signieren mit Hilfe von Hashfunktionen . . . . .	194
6.2 RSA-Signatur . . . . .	194
6.3 DSA-Signatur . . . . .	195
6.4 Public-Key-Zertifizierung . . . . .	195
6.4.1 Die Impersonalisierungsattacke . . . . .	195
6.4.2 X.509-Zertifikat . . . . .	197
Literaturverzeichnis . . . . .	198
<b>7 Elliptische Kurven</b>	<b>200</b>
7.1 Elliptische Kurven – ein effizienter Ersatz für RSA? . . . . .	200
7.2 Elliptische Kurven – Historisches . . . . .	202
7.3 Elliptische Kurven – Mathematische Grundlagen . . . . .	203
7.3.1 Gruppen . . . . .	203
7.3.2 Körper . . . . .	204
7.4 Elliptische Kurven in der Kryptographie . . . . .	206
7.5 Verknüpfung auf Elliptischen Kurven . . . . .	208
7.6 Sicherheit der Elliptischen-Kurven-Kryptographie: Das ECDLP . . . . .	211
7.7 Verschlüsseln und Signieren mit Hilfe Elliptischer Kurven . . . . .	212

7.7.1	Verschlüsselung . . . . .	212
7.7.2	Signatur-Erstellung . . . . .	213
7.7.3	Signatur-Verifikation . . . . .	213
7.8	Faktorisieren mit Elliptischen Kurven . . . . .	213
7.9	Implementierung Elliptischer Kurven zu Lehrzwecken . . . . .	215
7.9.1	CrypTool . . . . .	215
7.9.2	Sage . . . . .	215
7.10	Patentaspekte . . . . .	217
7.11	Elliptische Kurven im praktischen Einsatz . . . . .	217
	Literaturverzeichnis . . . . .	219
	Web-Links . . . . .	220
<b>8</b>	<b>Krypto 2020 — Perspektiven für langfristige kryptographische Sicherheit</b>	<b>221</b>
8.1	Verbreitete Verfahren . . . . .	221
8.2	Vorsorge für morgen . . . . .	222
8.3	Neue mathematische Probleme . . . . .	223
8.4	Neue Signaturen . . . . .	224
8.5	Quantenkryptographie – Ein Ausweg? . . . . .	224
8.6	Fazit . . . . .	224
	Literaturverzeichnis . . . . .	225
<b>A</b>	<b>Anhang</b>	<b>226</b>
A.1	CrypTool-Menübaum . . . . .	227
A.2	Autoren des CrypTool-Skripts . . . . .	229
A.3	Filme und belletristische Literatur mit Bezug zur Kryptographie, Kinderbücher mit einfachen Verschlüsselungen . . . . .	231
A.3.1	Für Erwachsene und Jugendliche . . . . .	231
A.3.2	Für Kinder und Jugendliche . . . . .	239
A.4	Lernprogramm Elementare Zahlentheorie . . . . .	241
A.5	Kurzeinführung in das Computeralgebrasystem Sage . . . . .	247
	<b>GNU Free Documentation License</b>	<b>256</b>
	<b>Abbildungsverzeichnis</b>	<b>264</b>
	<b>Tabellenverzeichnis</b>	<b>265</b>
	<b>Verzeichnis der Krypto-Verfahren</b>	<b>267</b>
	<b>Verzeichnis der Sage-Programmbeispiele</b>	<b>268</b>



# Vorwort zur 10. Auflage des CrypTool-Skripts

Ab dem Jahr 2000 wurde mit dem CrypTool-Paket auch ein Skript ausgeliefert, das die Mathematik einzelner Themen genauer, aber doch möglichst verständlich erläutern sollte.

Um auch hier die Möglichkeit zu schaffen, dass getrennte Entwickler/Autoren mitarbeiten können, wurden die Themen sinnvoll unterteilt und dafür jeweils eigenständig lesbare Kapitel geschrieben. In der anschließenden redaktionellen Arbeit wurden in TeX Querverweise ergänzt und Fußnoten hinzugefügt, die zeigen, an welchen Stellen man die entsprechenden Funktionen in CrypTool v1 aufrufen kann (vgl. den Menübaum im Anhang A.1). Natürlich gäbe es viel mehr Themen in Mathematik und Kryptographie, die man vertiefen könnte – deshalb ist diese Auswahl auch nur eine von vielen möglichen.

Der Erfolg des Internets hat zu einer verstärkten Forschung der damit verbundenen Technologien geführt, was auch im Bereich Kryptographie viele neue Erkenntnisse schaffte.

In dieser Ausgabe des Skripts wurden die TeX-Sourcen des Dokuments komplett überarbeitet, und etliche Themen korrigiert, ergänzt und auf den aktuellen Stand gebracht, z.B.:

- die größten Primzahlen (Kap. 3.4), neue Faktorisierungsrekorde (Kap. 4.11.4),
- Fortschritte bei der Kryptoanalyse von AES (Kap. 1.1.1) und
- die Auflistung, in welchen Filmen und Romanen Kryptographie eine wesentliche Rolle spielt (siehe Anhang A.3), und Kurioses zu Primzahlen (siehe 3.8.5).
- Neu hinzu gekommen sind z.B. der Absatz zu Benfords Gesetz und Primzahlen (Kap. 3.8.2), und der Anhang A.5 über das Computer-Algebra-System Sage. Sage wird mehr und mehr zum Standard-Open-Source CAS. Entsprechend wurden alle Codebeispiele, die vorher in PARI-GP und Mathematica geschrieben waren, durch Sage-Code ersetzt. Außerdem kamen dank Nguyen und Massierer etliche neue Code-Beispiele hinzu (vgl. auch Kap. 7.9.2).

Seit das Skript dem CrypTool-Paket in Version 1.2.01 das erste Mal beigelegt wurde, ist es mit fast jeder neuen Version von CrypTool ebenfalls erweitert und aktualisiert worden.

Herzlichen Dank an alle, die mit Ihrem großem Einsatz zum Erfolg und zur weiten Verbreitung dieses Projekts beigetragen haben. Danke auch an die Leser, die uns Feedback sandten.

Ich hoffe, dass viele Leser mit diesem Skript mehr Interesse an und Verständnis für dieses moderne und zugleich uralte Thema finden.

Bernhard Esslinger

Frankfurt, Januar 2010

# Einführung – Zusammenspiel von Skript und CrypTool

## Das Skript

Dieses Skript wird zusammen mit dem Open-Source-Programm CrypTool ausgeliefert.

Die Artikel dieses Skripts sind weitgehend in sich abgeschlossen und können auch unabhängig von CrypTool gelesen werden.

Während für das Verständnis der meisten Kapitel Abiturwissen ausreicht, erfordern Kapitel 5 (Moderne Kryptografie) und 7 (Elliptische Kurven) tiefere mathematische Kenntnisse.

Die Autoren haben sich bemüht, Kryptographie für eine möglichst breite Leserschaft darzustellen – ohne mathematisch unkorrekt zu werden. Dieser didaktische Anspruch ist am besten geeignet, die Awareness für die IT-Sicherheit und die Bereitschaft für den Einsatz standardisierter, moderner Kryptographie zu fördern.

## Das Programm CrypTool

CrypTool ist ein Lernprogramm mit umfangreicher Online-Hilfe, mit dem Sie unter einer einheitlichen Oberfläche kryptographische Verfahren anwenden und analysieren können.

Inzwischen wird CrypTool weltweit in Ausbildung und Lehre eingesetzt und von mehreren Hochschulen mit weiterentwickelt.

## Dank

An dieser Stelle möchte ich explizit folgenden Personen danken, die bisher in ganz besonderer Weise zu CrypTool beigetragen haben. Ohne ihre besonderen Fähigkeiten und ihr großes Engagement wäre CrypTool nicht, was es heute ist:

- Hr. Henrik Koy
- Hr. Jörg-Cornelius Schneider
- Hr. Florian Marchal
- Dr. Peer Wichmann
- Mitarbeiter von Prof. Claudia Eckert, Prof. Johannes Buchmann und Prof. Torben Weis.

Auch allen hier nicht namentlich Genannten ganz herzlichen Dank für das (meist in der Freizeit) geleistete Engagement.

Bernhard Esslinger

Frankfurt, November 2009

# Kapitel 1

## Verschlüsselungsverfahren

(Bernhard Esslinger, Jörg-Cornelius Schneider, Mai 1999; Updates: Dez. 2001, Feb. 2003, Juni 2005, Juli 2007, Januar 2010)

*Indisches Sprichwort:*

Erkläre es mir, ich werde es vergessen.

Zeige es mir, ich werde es vielleicht behalten.

Lass es mich tun, und ich werde es können.

Dieses Kapitel soll einen eher beschreibenden Einstieg bieten und die Grundlagen ohne Verwendung von allzuviel Mathematik vermitteln.

Sinn der Verschlüsselung ist es, Daten so zu verändern, dass nur ein autorisierter Empfänger in der Lage ist, den Klartext zu rekonstruieren. Das hat den Vorteil, dass verschlüsselte Daten offen übertragen werden können und trotzdem keine Gefahr besteht, dass ein Angreifer die Daten unberechtigt lesen kann. Der autorisierte Empfänger ist im Besitz einer geheimen Information, des sogenannten Schlüssels, die es ihm erlaubt, die Daten zu entschlüsseln, während sie jedem anderen verborgen bleiben.

Es gibt ein beweisbar sicheres Verschlüsselungsverfahren, das *One-Time-Pad*. Dieses weist allerdings einige praktische Nachteile auf (der verwendete Schlüssel muss zufällig gewählt werden und mindestens so lang wie die zu schützende Nachricht sein), so dass es außer in geschlossenen Umgebungen, zum Beispiel beim heißen Draht zwischen Moskau und Washington, kaum eine Rolle spielt.

Für alle anderen Verfahren gibt es (theoretische) Möglichkeiten, sie zu brechen. Bei guten Verfahren sind diese jedoch so aufwändig, dass sie praktisch nicht durchführbar sind und diese Verfahren als (praktisch) sicher angesehen werden können.

Einen sehr guten Überblick zu den Verfahren bietet das Buch von Bruce Schneier [Schneier1996]. Grundsätzlich unterscheidet man zwischen symmetrischen und asymmetrischen Verfahren zur Verschlüsselung.

## 1.1 Symmetrische Verschlüsselung<sup>1</sup>

Bei der *symmetrischen* Verschlüsselung müssen Sender und Empfänger über einen gemeinsamen (geheimen) Schlüssel verfügen, den sie vor Beginn der eigentlichen Kommunikation ausgetauscht haben. Der Sender benutzt diesen Schlüssel, um die Nachricht zu verschlüsseln und der Empfänger, um diese zu entschlüsseln.

Alle klassischen Verfahren sind von diesem Typ. Beispiele dazu finden Sie im Programm CrypTool, im Kapitel 2 („Papier- und Bleistift-Verschlüsselungsverfahren“) in diesem Skript oder in [Nichols1996]. Im folgenden wollen wir jedoch nur die moderneren Verfahren betrachten.

Vorteile von symmetrischen Algorithmen sind die hohe Geschwindigkeit, mit denen Daten ver- und entschlüsselt werden. Ein Nachteil ist das Schlüsselmanagement. Um miteinander vertraulich kommunizieren zu können, müssen Sender und Empfänger vor Beginn der eigentlichen Kommunikation über einen sicheren Kanal einen Schlüssel ausgetauscht haben. Spontane Kommunikation zwischen Personen, die sich vorher noch nie begegnet sind, scheint so nahezu unmöglich. Soll in einem Netz mit  $n$  Teilnehmern jeder mit jedem zu jeder Zeit spontan kommunizieren können, so muss jeder Teilnehmer vorher mit jedem anderen der  $n - 1$  Teilnehmer einen Schlüssel ausgetauscht haben. Insgesamt müssen also  $n(n - 1)/2$  Schlüssel ausgetauscht werden.

Das bekannteste symmetrische Verschlüsselungsverfahren ist der DES-Algorithmus. Der DES-Algorithmus ist eine Entwicklung von IBM in Zusammenarbeit mit der National Security Agency (NSA). Er wurde 1975 als Standard veröffentlicht. Trotz seines relativ hohen Alters ist jedoch bis heute kein „effektiver“ Angriff auf ihn gefunden worden. Der effektivste Angriff besteht aus dem Durchprobieren (fast) aller möglichen Schlüssel, bis der richtige gefunden wird (*brute-force-attack*). Aufgrund der relativ kurzen Schlüssellänge von effektiv 56 Bits (64 Bits, die allerdings 8 Paritätsbits enthalten), sind in der Vergangenheit schon mehrfach mit dem DES verschlüsselte Nachrichten gebrochen worden, so dass er heute als nur noch bedingt sicher anzusehen ist. Symmetrische Alternativen zum DES sind zum Beispiel die Algorithmen IDEA oder Triple-DES.

Hohe Aktualität besitzt das symmetrische AES-Verfahren: Der dazu gehörende Rijndael-Algorithmus wurde am 2. Oktober 2000 zum Gewinner der AES-Ausschreibung erklärt und ist damit Nachfolger des DES-Verfahrens.

Näheres zum AES-Algorithmus und den AES-Kandidaten der letzten Runde finden Sie in der CrypTool-Online-Hilfe<sup>2</sup>.

### 1.1.1 Neue Ergebnisse zur Kryptoanalyse von AES

Im Anschluss finden Sie einige Informationen, die das AES-Verfahren in letzter Zeit in Zweifel zogen – unserer Ansicht nach aber (noch) unbegründet. Die folgenden Informationen beruhen auf den unten angegebenen Originalarbeiten und auf [Wobst-iX2002] und [Luks-DuD2002].

Der AES bietet mit einer Mindestschlüssellänge von 128 Bit gegen Brute-Force-Angriffe auch auf längere Sicht genügend Sicherheit – es sei denn, es stünden entsprechend leistungsfähige

<sup>1</sup>Mit CrypTool können Sie über das Menü **Ver-/Entschlüsseln \ Symmetrisch (modern)** folgende modernen symmetrischen Verschlüsselungsverfahren ausführen:

IDEA, RC2, RC4, DES (ECB), DES (CBC), Triple-DES (ECB), Triple-DES (CBC), MARS (AES-Kandidat), RC6 (AES-Kandidat), Serpent (AES-Kandidat), Twofish (AES-Kandidat), Rijndael (offizielles AES-Verfahren).

<sup>2</sup>CrypTool-Online-Hilfe: Das Stichwort **AES** im Index führt auf die 3 Hilfeseiten: **AES-Kandidaten**, **Der AES-Gewinner Rijndael** und **Der AES-Algorithmus Rijndael**.



Quantencomputer zur Verfügung. Der AES war immun gegen alle bis dahin bekannten Kryptoanalyse-Verfahren, die vor allem auf statistischen Überlegungen beruhen und auf DES angewandt wurden: man konstruiert aus Geheim- und Klartextpaaren Ausdrücke, die sich nicht rein zufällig verhalten, sondern Rückschlüsse auf den verwendeten Schlüssel zulassen. Dazu waren aber unrealistische Mengen an abgehörten Daten nötig.

Bei Kryptoverfahren bezeichnet man solche Kryptoanalyseverfahren als „akademischen Erfolg“ oder als „kryptoanalytischen Angriff“, da sie theoretisch schneller sind als das Durchprobieren aller Schlüssel, das beim Brute-Force-Angriff verwendet wird. Im Fall des AES mit maximaler Schlüssellänge (256 Bit) braucht die erschöpfende Schlüsselsuche im Durchschnitt  $2^{255}$  Verschlüsselungsoperationen. Ein kryptoanalytischer Angriff muss diesen Wert unterbieten. Als aktuell gerade noch praktikabel (z.B. für einen Geheimdienst) gilt ein Aufwand von  $2^{75}$  bis  $2^{90}$  Verschlüsselungsoperationen.

Eine neue Vorgehensweise wurde in der Arbeit von Ferguson, Schroeppel und Whiting im Jahre 2001 [Ferguson2001] beschrieben: sie stellten AES als geschlossene Formel (in der Form einer Art Kettenbruch) dar, was aufgrund seiner „relativ“ übersichtlichen Struktur gelang. Da diese Formel aus rund 1 Billionen Summanden besteht, taugt sie zunächst nicht für die Kryptoanalyse. Dennoch war die wissenschaftliche Neugier geweckt. Bereits bekannt war, dass sich der 128-Bit-AES als ein überbestimmtes System von rund 8000 quadratischen Gleichungen (über algebraischen Zahlkörpern) mit rund 1600 Variablen (einige Unbekannte sind die Schlüsselbits) darstellen lässt – solch große Gleichungssysteme sind praktisch nicht lösbar. Dieses Gleichungssystem ist relativ dünn besetzt („sparse“), d.h. von den insgesamt etwa 1.280.000 möglichen quadratischen Termen tauchen nur relativ wenige überhaupt im Gleichungssystem auf.

Die Mathematiker Courois und Pieprzyk [Courtois2002] veröffentlichten 2002 eine Arbeit, die in der Krypto-Szene stark diskutiert wird: Sie entwickelten das auf der Eurocrypt 2000 von Shamir et al. vorgestellte XL-Verfahren (eXtended Linearization) weiter zum XSL-Verfahren (eXtended Sparse Linearization). Das XL-Verfahren ist eine heuristische Technik, mit der es manchmal gelingt, große nicht-lineare Gleichungssysteme zu lösen und die bei der Analyse eines asymmetrischen Algorithmus (HFE) angewandt wurde. Die Innovation von Courois und Pieprzyk war, das XL-Verfahren auf symmetrische Verfahren anzuwenden: das XSL-Verfahren kann auf spezielle Gleichungssysteme angewandt werden. Damit könnte ein 256-Bit-AES-Verfahren in rund  $2^{230}$  Schritten geknackt werden. Dies ist zwar immer noch ein rein akademischer Angriff, aber er ist richtungsweisend für eine ganze Klasse von Blockchiffren. Das generelle Problem mit diesem Angriff besteht darin, dass man bisher nicht angeben kann, unter welchen Umständen er zum Erfolg führt: die Autoren geben in ihrer Arbeit notwendige Bedingungen an; es ist nicht bekannt, welche Bedingungen hinreichend sind. Neu an diesem Angriff war erstens, dass dieser Angriff nicht auf Statistik, sondern auf Algebra beruhte. Dadurch erscheinen Angriffe möglich, die nur geringe Mengen von Geheimtext brauchen. Zweitens steigt damit die Sicherheit eines Produktalgorithmus<sup>3</sup> nicht mehr exponentiell mit der Rundenzahl.

---

<sup>3</sup>Ein Geheimtext kann selbst wieder Eingabe für eine Chiffrierung sein. Eine Mehrfachverschlüsselung (cascade cipher) entsteht aus einer Komposition von mehreren Verschlüsselungstransformationen. Die Gesamtchiffrierung wird Produktalgorithmus oder Kaskadenalgorithmus genannt (manchmal ist die Namensgebung abhängig davon, ob die verwendeten Schlüssel statistisch abhängig oder unabhängig sind).

Nicht immer wird die Sicherheit eines Verfahrens durch Produktbildung erhöht.

Dieses Vorgehen wird auch *innerhalb* moderner Algorithmen angewandt: Sie kombinieren in der Regel einfache und, für sich genommen, kryptologisch relativ unsichere Einzelschritte in mehreren Runden zu einem leistungsfähigen Gesamtverfahren. Die meisten modernen Blockchiffrierungen (z.B. DES, IDEA) sind Produktalgorithmen.

Als Mehrfachverschlüsselung wird auch das Hintereinanderschalten desselben Verfahrens mit verschiedenen Schlüsseln wie bei Triple-DES bezeichnet.

Aktuell wird sehr intensiv auf diesem Gebiet geforscht: z.B. stellten Murphy und Robshaw auf der Crypto 2002 ein Papier vor [Robshaw2002a], das die Kryptoanalyse drastisch verbessern könnte: der Aufwand für 128-Bit-Schlüssel wurde auf  $2^{100}$  geschätzt, indem sie AES als Spezialfall eines Algorithmus BES (Big Encryption System) darstellten, der eine besonders „runde“ Struktur hat. Aber auch  $2^{100}$  Rechenschritte liegen jenseits dessen, was praktisch in absehbarer Zeit realisierbar ist. Bei 256 Bit Schlüssellänge schätzen die Autoren den Aufwand für den XSL-Angriff auf  $2^{200}$  Operationen.

Weitere Details dazu stehen unter:

<http://www.cryptosystem.net/aes>

<http://www.minrank.org/aes/>

Für 256-AES wäre der Angriff ebenfalls viel besser als Brute-Force, aber auch noch weiter außerhalb der Reichweite realisierbarer Rechenpower.

Die Diskussion ist z.Zt. sehr kontrovers: Don Coppersmith (einer der DES-Erfinder) z.B. bezweifelt die generelle Durchführbarkeit des Angriffs: XLS liefere für AES gar keine Lösung [Coppersmith2002]. Dann würde auch die Optimierung von Murphy und Robshaw [Robshaw2002b] nicht greifen.

2009 veröffentlichten Biryukov und Khovratovich [Biryukov2009] einen weiteren theoretischen Angriff auf AES, der sich anderer Techniken bedient, als die oben vorgestellten. Sie verwenden Methoden aus der Kryptoanalyse von Hashfunktionen (lokale Kollisionen und das Boomerang-Verfahren) und konstruierten daraus einen Related-Key-Angriff auf AES-256. D. h. der Angreifer muss nicht nur in der Lage sein, beliebige Daten zu verschlüsseln (chosen plain text), sondern auch den ihm unbekannten Schlüssel manipulieren können (related key).

Unter diesen Angriffs-Voraussetzungen reduziert der Angriff den Aufwand, einen AES-256-Schlüssel zu ermitteln, auf  $2^{119}$  Zeit und  $2^{77}$  Speicher. Bei AES-192 ist der Angriff noch weniger praktikabel, für AES-128 geben die Autoren keinen Angriff an.

### 1.1.2 Aktueller Stand der Brute-Force-Angriffe auf symmetrische Verfahren (RC5)

Anhand der Blockchiffre RC5 kann der aktuelle Stand von Brute-Force-Angriffen auf symmetrische Verschlüsselungsverfahren gut erläutert werden.

Als Brute-Force-Angriff (exhaustive search, trial-and-error) bezeichnet man das vollständige Durchsuchen des Schlüsselraumes: dazu müssen keine besonderen Analysetechniken eingesetzt werden. Stattdessen wird der Ciphertext mit allen möglichen Schlüsseln des Schlüsselraums entschlüsselt und geprüft, ob der resultierende Text einen sinnvollen Klartext ergibt. Bei einer Schlüssellänge von 64 Bit sind dies maximal  $2^{64} = 18.446.744.073.709.551.616$  oder rund 18 Trillionen zu überprüfende Schlüssel<sup>4</sup>.

Um zu zeigen, welche Sicherheit bekannte symmetrische Verfahren wie DES, Triple-DES oder RC5 haben, veranstaltet z.B. RSA Security sogenannte Cipher-Challenges<sup>5</sup>. Unter kontrollierten Bedingungen werden Preise ausgelobt, um Ciphertexte (verschlüsselt mit verschiedenen

---

<sup>4</sup>Mit CrypTool können Sie über das Menü **Analyse \ Symmetrische Verschlüsselung (modern)** ebenfalls Brute-Force-Analysen von modernen symmetrischen Verfahren durchführen (unter der schwächsten aller möglichen Annahmen: der Angreifer kennt nur den Geheimtext, er führt also einen Ciphertext only-Angriff durch). Um in einer sinnvollen Zeit auf einem Einzel-PC ein Ergebnis zu erhalten, sollten Sie nicht mehr als 20 Bit des Schlüssels als unbekannt kennzeichnen.

<sup>5</sup><http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html>

Im Mai 2007 meldete RSA, dass sie die Korrektheit der bis dahin nicht gelösten RC5-72 Challenge nicht bestätigen werden.

Verfahren und verschiedenen Schlüssellängen) zu entschlüsseln und den symmetrischen Schlüssel zu ermitteln. Damit werden theoretische Resultate praktisch bestätigt.

Dass das „alte“ Standard-Verfahren DES mit der fixen Schlüssellänge von 56 Bit nicht mehr sicher ist, wurde schon im Januar 1999 von der Electronic Frontier Foundation (EFF) demonstriert, als sie mit Deep Crack eine DES-verschlüsselte Nachricht in weniger als einem Tag knackten<sup>6</sup>.

Der aktuelle Rekord für starke symmetrische Verfahren liegt bei 64 Bit langen Schlüsseln. Dazu wurde das Verfahren RC5 benutzt, eine Blockchiffre mit variabler Schlüssellänge.

Die RC5-64 Challenge wurde im September 2002 nach 5 Jahren vom distributed.net-Team gelöst<sup>7</sup>. Insgesamt arbeiteten 331.252 Personen gemeinsam über das Internet zusammen.<sup>8</sup> Getestet wurden rund 15 Trillionen Schlüssel, bis der richtige Schlüssel gefunden wurde.

Damit ist gezeigt, dass symmetrische Verfahren (auch wenn sie keinerlei kryptographische Schwächen haben) mit 64 Bit langen Schlüsseln keine geeigneten Verfahren mehr sind, um sensible Daten länger geheim zu halten.

Cipher-Challenges gibt es auch für asymmetrische Verfahren (siehe Kapitel 4.11.4)<sup>9</sup>.

## 1.2 Asymmetrische Verschlüsselung<sup>10</sup>

Bei der *asymmetrischen* Verschlüsselung hat jeder Teilnehmer ein persönliches Schlüsselpaar, das aus einem *geheimen* und einem *öffentlichen* Schlüssel besteht. Der öffentliche Schlüssel wird, der Name deutet es an, öffentlich bekanntgemacht, zum Beispiel in einem Schlüsselverzeichnis im Internet.

Möchte Alice<sup>11</sup> mit Bob kommunizieren, so sucht sie Bobs öffentlichen Schlüssel aus dem Verzeichnis und benutzt ihn, um ihre Nachricht an ihn zu verschlüsseln. Diesen verschlüsselten Text schickt sie dann an Bob, der mit Hilfe seines geheimen Schlüssels den Text wieder entschlüsseln kann. Da einzig Bob Kenntnis von seinem geheimen Schlüssel hat, ist auch nur er in der Lage, an ihn adressierte Nachrichten zu entschlüsseln. Selbst Alice als Absenderin der Nachricht kann aus der von ihr versandten (verschlüsselten) Nachricht den Klartext nicht wieder herstellen. Natürlich muss sichergestellt sein, dass man aus dem öffentlichen Schlüssel nicht auf den geheimen Schlüssel schließen kann.

Veranschaulichen kann man sich ein solches Verfahren mit einer Reihe von einbruchssicheren Briefkästen. Wenn ich eine Nachricht verfasst habe, so suche ich den Briefkasten mit dem Namensschild des Empfängers und werfe den Brief dort ein. Danach kann ich die Nachricht selbst nicht mehr lesen oder verändern, da nur der legitime Empfänger im Besitz des Schlüssels für den Briefkasten ist.

---

<sup>6</sup><http://www.rsasecurity.com/rsalabs/challenges/des3/index.html>

<sup>7</sup><http://distributed.net/pressroom/news-20020926.html>

<sup>8</sup>Eine Übersicht über die aktuellen Projekte zum verteilten Rechnen finden Sie unter:  
<http://distributedcomputing.info/>

<sup>9</sup>Ein weites Spektrum einfacher und komplexer Krypto-Aufgaben bietet der Wettbewerb „Mystery Twister C3“.

<sup>10</sup>Mit CrypTool können Sie über das Menü **Ver-/Entschlüsseln** \ **Asymmetrisch** mit RSA ver- und entschlüsseln. In beiden Fällen müssen Sie ein RSA-Schlüsselpaar auswählen. Nur bei der Entschlüsselung wird der geheime RSA-Schlüssel benötigt: deshalb wird nur hier die PIN abgefragt.

<sup>11</sup>Zur Beschreibung kryptographischer Protokolle werden den Teilnehmern oft Namen gegeben (vergleiche [Schneier1996, S. 23]). Alice und Bob führen alle allgemeinen 2-Personen-Protokolle durch, wobei Alice dies initiiert und Bob antwortet. Die Angreifer werden als Eve (eavesdropper = passiver Lauscher) und Mallory (malicious active attacker = böswilliger, aktiver Abgreifer) bezeichnet.

Vorteil von asymmetrischen Verfahren ist das einfache Schlüsselmanagement. Betrachten wir wieder ein Netz mit  $n$  Teilnehmern. Um sicherzustellen, dass jeder Teilnehmer jederzeit eine verschlüsselte Verbindung zu jedem anderen Teilnehmer aufbauen kann, muss jeder Teilnehmer ein Schlüsselpaar besitzen. Man braucht also  $2n$  Schlüssel oder  $n$  Schlüsselpaare. Ferner ist im Vorfeld einer Übertragung kein sicherer Kanal notwendig, da alle Informationen, die zur Aufnahme einer vertraulichen Kommunikation notwendig sind, offen übertragen werden können. Hier ist lediglich auf die Unverfälschtheit (Integrität und Authentizität) des öffentlichen Schlüssels zu achten. Nachteil: Im Vergleich zu symmetrischen Verfahren sind reine asymmetrische Verfahren jedoch um ein Vielfaches langsamer.

Das bekannteste asymmetrische Verfahren ist der RSA-Algorithmus<sup>12</sup>, der nach seinen Entwicklern Ronald Rivest, Adi Shamir und Leonard Adleman benannt wurde. Der RSA-Algorithmus wurde 1978 veröffentlicht. Das Konzept der asymmetrischen Verschlüsselung wurde erstmals von Whitfield Diffie und Martin Hellman im Jahre 1976 vorgestellt. Heute spielen auch die Verfahren nach ElGamal eine bedeutende Rolle, vor allem die Schnorr-Varianten im DSA (Digital Signature Algorithm).

### 1.3 Hybridverfahren<sup>13</sup>

Um die Vorteile von symmetrischen und asymmetrischen Techniken gemeinsam nutzen zu können, werden (zur Verschlüsselung) in der Praxis meist Hybridverfahren verwendet.

Hier werden die Mengen-Daten mittels symmetrischer Verfahren verschlüsselt: Der Schlüssel ist ein vom Absender zufällig<sup>14</sup> generierter geheimer Sitzungsschlüssel (session key), der nur für diese Nachricht verwendet wird.

Anschließend wird dieser Sitzungsschlüssel mit Hilfe des asymmetrischen Verfahrens verschlüsselt und zusammen mit der Nachricht an den Empfänger übertragen.

Der Empfänger kann den Sitzungsschlüssel mit Hilfe seines geheimen Schlüssels bestimmen und mit diesem dann die Nachricht entschlüsseln.

Auf diese Weise profitiert man von dem bequemen Schlüsselmanagement asymmetrischer Verfahren (mit öffentlichem und privatem Schlüssel), und man profitiert von der Schnelligkeit symmetrischer Verfahren, um große Datenmengen zu verschlüsseln (mit den geheimen Schlüsseln).

---

<sup>12</sup>RSA wird in diesem Skript ab Kapitel 4.10 ausführlich beschrieben. Das RSA-Kryptosystem kann mit CrypTool über das Menü **Einzelverfahren \ RSA-Kryptosystem \ RSA-Demo** in vielen Variationen nachvollzogen werden. Die aktuellen Forschungsergebnisse im Umfeld von RSA werden in Kapitel 4.11 beschrieben.

<sup>13</sup>In CrypTool finden Sie dieses Verfahren über das Menü **Ver-/Entschlüsseln \ Hybrid**. Dabei können Sie die einzelnen Schritte und ihre Abhängigkeiten mit konkreten Zahlen nachvollziehen. Die Variante mit RSA als asymmetrischem Verfahren ist graphisch visualisiert; die Variante mit ECC nutzt die Standard-Dialoge. In beiden Fällen wird AES als symmetrisches Verfahren eingesetzt.

<sup>14</sup>Die Erzeugung zufälliger Zahlen ist ein wichtiger Bestandteil kryptographisch sicherer Verfahren. Mit CrypTool können Sie über das Menü **Einzelverfahren \ Zufallsdaten erzeugen** verschiedene Zufallszahlengeneratoren ausprobieren. Über das Menü **Analyse \ Zufallsanalyse** können Sie verschiedene Testverfahren für Zufallsdaten auf binäre Dokumente anwenden.

Bisher konzentriert sich CrypTool auf kryptographisch starke Pseudozufallszahlengeneratoren. Nur im Secude-Generator wird eine „echte“ Zufallsquelle einbezogen.

*IETF*<sup>15</sup>:

Ein altes Sprichwort, angeblich geprägt von der US National Security Agency (NSA), sagt:  
“Angriffe werden immer besser, niemals schlechter.”  
“Attacks always get better; they never get worse.”

## 1.4 Chiffren und Kryptoanalyse für Lehrzwecke

Verglichen mit den auf der Zahlentheorie beruhenden Public-Key-Verschlüsselungsverfahren wie RSA, ist die Struktur von AES und den meisten anderen modernen symmetrischen Verschlüsselungsverfahren sehr komplex und kann nicht so einfach wie RSA erklärt werden.

Deshalb wurden zu Lehrzwecken einige vereinfachte Varianten moderner symmetrischer Verfahren wie DES, IDEA oder AES entwickelt, um Einsteigern die Möglichkeit zu geben, Ver- und Entschlüsselung von Hand zu lernen und ein besseres Verständnis zu gewinnen, wie die Algorithmen im Detail funktionieren. Diese vereinfachten Varianten helfen auch, die entsprechenden Kryptoanalyse-Methoden zu verstehen und anzuwenden.

Eine dieser Varianten ist z.B. S-AES (Simplified-AES) von Prof. Ed Schaefer und seinen Studenten [Musa-etal2003]<sup>16</sup>. Eine andere Variante ist Mini-AES [Phan2002] (siehe das Kapitel 1.6.1 “Mini-AES”):

- Edward F. Schaefer: *A Simplified Data Encryption Standard Algorithm* [Schaefer1996].
- Raphael Chung-Wei Phan: *Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students* [Phan2002].
- Raphael Chung-Wei Phan: *Impossible differential cryptanalysis of Mini-AES* [Phan2003].
- Mohammad A. Musa, Edward F. Schaefer, Stephen Wedig: *A simplified AES algorithm and its linear and differential cryptanalyses* [Musa-etal2003].
- Nick Hoffman: *A SIMPLIFIED IDEA ALGORITHM* [Hoffman2006].
- S. Davod. Mansoori, H. Khaleghi Bizaki: *On the vulnerability of Simplified AES Algorithm Against Linear Cryptanalysis* [Mansoori-etal2007].

## 1.5 Weitere Details

Neben den anderen Kapiteln in diesem Skript, der umfangreichen Fachliteratur und vielen Stellen im Internet enthält auch die Online-Hilfe von CrypTool sehr viele weitere Informationen zu den einzelnen symmetrischen und asymmetrischen Verschlüsselungsverfahren.

<sup>15</sup><http://tools.ietf.org/html/rfc4270>

<sup>16</sup>Siehe auch die beiden namensgleichen Artikel “Devising a Better Way to Teach and Learn the Advanced Encryption Standard” unter den Universitäts-News bei  
- <http://math.scu.edu/~eschaefer/getfile.pdf>  
- <http://www.scu.edu/cas/research/cryptography.cfm>

## 1.6 Anhang: Beispiele mit Sage

Der folgende Abschnitt enthält Sage Source-Code, der sich auf den Inhalt des Kapitels 1 („Verschlüsselungsverfahren“) bezieht.

Weitere Details zu in Sage enthaltenen Kryptoverfahren (z.B. S-DES) finden sich z.B. in der Diplomarbeit von Minh Van Nguyen [Nguyen2009].

### 1.6.1 Mini-AES

Das Sage-Modul `crypto/block_cipher/miniaes.py` enthält den Mini-AES, damit Studenten die Funktionsweise moderner Blockchiffren untersuchen können.

Mini-AES, ursprünglich vorgestellt von [Phan2002], ist eine vereinfachte Variante des Advanced Encryption Standard (AES) für Ausbildungszwecke.

Wie man Mini-AES benutzt, ist ausführlich in der Sage Reference-Page beschrieben:  
[http://www.sagemath.org/doc/reference/sage/crypto/block\\_cipher/miniaes.html](http://www.sagemath.org/doc/reference/sage/crypto/block_cipher/miniaes.html).

Das folgende Sage Code-Beispiel von den Release-Notes von Sage 4.1<sup>17</sup> ruft diese Implementierung des Mini-AES auf.

Weiterer Beispiel-Code zum Mini-AES findet sich in [Nguyen2009, Kap. 6.5 und Anhang D].

---

<sup>17</sup>Siehe <http://mvngu.wordpress.com/2009/07/12/sage-4-1-released/>

---

### Sage-Beispiel 1.1 Ver- und Entschlüsselung mit dem Mini-AES

---

```
# We can encrypt a plaintext using Mini-AES as follows:
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: K = FiniteField(16, "x")
sage: MS = MatrixSpace(K, 2, 2)
sage: P = MS([K("x^3 + x"), K("x^2 + 1"), K("x^2 + x"), K("x^3 + x^2")]); P

[ x^3 + x  x^2 + 1]
[ x^2 + x x^3 + x^2]
sage: key = MS([K("x^3 + x^2"), K("x^3 + x"), K("x^3 + x^2 + x"), K("x^2 + x + 1")]); key

[ x^3 + x^2  x^3 + x]
[x^3 + x^2 + x  x^2 + x + 1]
sage: C = maes.encrypt(P, key); C

[ x  x^2 + x]
[x^3 + x^2 + x  x^3 + x]

# Here is the decryption process:
sage: plaintext = maes.decrypt(C, key)
sage: plaintext == P
True

# We can also work directly with binary strings:
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: bin = BinaryStrings()
sage: key = bin.encoding("KE"); key
0100101101000101
sage: P = bin.encoding("Encrypt this secret message!")
sage: C = maes(P, key, algorithm="encrypt")
sage: plaintext = maes(C, key, algorithm="decrypt")
sage: plaintext == P
True

# Or work with integers n such that 0 <= n <= 15:
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: P = [n for n in xrange(16)]; P
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
sage: key = [2, 3, 11, 0]; key
[2, 3, 11, 0]
sage: P = maes.integer_to_binary(P)
sage: key = maes.integer_to_binary(key)
sage: C = maes(P, key, algorithm="encrypt")
sage: plaintext = maes(C, key, algorithm="decrypt")
sage: plaintext == P
True
```

---

# Literaturverzeichnis

- [Biryukov2009] Alex Biryukov, Dmitry Khovratovich,  
*Related-key Cryptanalysis of the Full AES-192 and AES-256*,  
2009  
<http://eprint.iacr.org/2009/317>
- [Coppersmith2002] Don Coppersmith,  
*Re: Impact of Courtois and Pieprzyk results*,  
19.09.2002, in den „AES Discussion Groups“ unter  
<http://aes.nist.gov/aes/>
- [Courtois2002] Nicolas Courtois, Josef Pieprzyk,  
*Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*,  
received 10 Apr 2002, last revised 9 Nov 2002.  
Eine davon verschiedene „compact version“ des ersten XSL-Angriffs wurde auf der  
Asiacrypt im Dezember 2002 veröffentlicht.  
<http://eprint.iacr.org/2002/044>
- [Ferguson2001] Niels Ferguson, Richard Schroepel, Doug Whiting,  
*A simple algebraic representation of Rijndael*, Draft 1. Mai 2001,  
<http://www.xs4all.nl/~vorpel/pubs/rdalgeq.html>
- [Hoffman2006] Nick Hoffman,  
*A SIMPLIFIED IDEA ALGORITHM*,  
2006  
<http://www.nku.edu/~christensen/simplified%20IDEA%20algorithm.pdf>
- [Lucks-DuD2002] Stefan Lucks, Rüdiger Weis,  
*Neue Ergebnisse zur Sicherheit des Verschlüsselungsstandards AES*,  
in DuD 12/2002.
- [Mansoori-et al2007] S. Davod. Mansoori, H. Khaleghi Bizaki,  
*On the vulnerability of Simplified AES Algorithm Against Linear Cryptanalysis*,  
In: IJCSNS International Journal of Computer Science and Network Security, Vol.7  
No.7, Juli 2007, Seite 257-263.  
Siehe auch:  
[http://paper.ijcsns.org/07\\_book/200707/20070735.pdf](http://paper.ijcsns.org/07_book/200707/20070735.pdf)
- [Musa-et al2003] Mohammad A. Musa, Edward F. Schaefer, Stephen Wedig,  
*A simplified AES algorithm and its linear and differential cryptanalyses*,  
In: Cryptologia 17 (2), April 2003, Seite 148-177.  
Siehe auch:



- [http://findarticles.com/p/articles/mi\\_qa3926/is\\_200304/ai\\_n9181658](http://findarticles.com/p/articles/mi_qa3926/is_200304/ai_n9181658)  
<http://www.rose-hulman.edu/~holden/Preprints/s-aes.pdf>  
<http://math.scu.edu/~eschaefer/> Ed Schaefer's Homepage
- [Nguyen2009] Minh Van Nguyen,  
*Exploring Cryptography Using the Sage Computer Algebra System*,  
 Victoria University, 2009,  
 Siehe Sage publications <http://www.sagemath.org/library-publications.html>,  
<http://www.sagemath.org/files/thesis/nguyen-thesis-2009.pdf>,  
<http://sites.google.com/site/nguyenminh2/honours-thesis-2009.pdf>
- [Nichols1996] Randall K. Nichols,  
*Classical Cryptography Course, Volume 1 and 2*,  
 Aegean Park Press 1996;  
 oder in 12 Lektionen online unter  
<http://www.fortunecity.com/skyscraper/coding/379/lesson1.htm>
- [Phan2002] Raphael Chung-Wei Phan,  
*Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students*,  
 In: Cryptologia 26 (4), 2002, Seite 283-306.
- [Phan2003] Raphael Chung-Wei Phan,  
*Impossible differential cryptanalysis of Mini-AES*,  
 In: Cryptologia, Oct 2003  
[http://findarticles.com/p/articles/mi\\_qa3926/is\\_200310/ai\\_n9311721/](http://findarticles.com/p/articles/mi_qa3926/is_200310/ai_n9311721/)
- [Robshaw2002a] S.P. Murphy, M.J.B. Robshaw,  
*Essential Algebraic Structure within the AES*,  
 5. Juni 2002, auf der Crypto 2002,  
<http://www.isg.rhul.ac.uk/~mrobshaw/rijndael/rijndael.html>
- [Robshaw2002b] S.P. Murphy, M.J.B. Robshaw,  
*Comments on the Security of the AES and the XSL Technique*,  
 26. September 2002,  
<http://www.isg.rhul.ac.uk/~mrobshaw/rijndael/rijndael.html>
- [Schaefer1996] Edward F. Schaefer,  
*A Simplified Data Encryption Standard Algorithm*,  
 In: Cryptologia 20 (1), 1996, Seite 77-84
- [Schmeh2009] Klaus Schmeh,  
*Kryptografie – Verfahren, Protokolle, Infrastrukturen*,  
 dpunkt.verlag, 4. aktualisierte und erweiterte Auflage 2009.  
 Sehr gut lesbares, aktuelles und umfangreiches Buch über Kryptographie. Geht auch  
 auf praktische Probleme, wie Standardisierung oder real existierende Software, ein.
- [Schneier1996] Bruce Schneier,  
*Applied Cryptography, Protocols, Algorithms, and Source Code in C*,  
 Wiley 1994, 2nd edition 1996.
- [Stallings2006] William Stallings,  
*Cryptography and Network Security*,  
 Prentice Hall 2006,  
<http://williamstallings.com/>

- [Stamp2007] Mark Stamp, Richard M. Low,  
*Applied Cryptanalysis: Breaking Ciphers in the Real World*,  
Wiley-IEEE Press 2007,  
<http://cs.sjsu.edu/faculty/stamp/crypto/>
- [Swenson2008] Christopher Swenson,  
*Modern Cryptanalysis: Techniques for Advanced Code Breaking*,  
Wiley 2008.
- [Wobst-iX2002] Reinhard Wobst,  
*Angekratzt - Kryptoanalyse von AES schreitet voran*,  
in iX 12/2002,  
und der Leserbrief dazu von Johannes Merkle in der iX 2/2003.

# Web-Links

1. Das AES-/Rijndael-Kryptosystem  
<http://www.cryptosystem.net/aes>  
<http://www.minrank.org/aes/>
2. „AES Discussion Groups” beim NIST  
<http://aes.nist.gov/aes>
3. distributed.net: „RC5-64 has been solved”  
<http://distributed.net/pressroom/news-20020926.html>
4. RSA: „The RSA Secret Key Challenge”  
<http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html>
5. RSA: „DES Challenge”  
<http://www.rsasecurity.com/rsalabs/challenges/des3/index.html>
6. Weiterführende Links auf der CrypTool-Homepage  
<http://www.cryptool.de>

## Kapitel 2

# Papier- und Bleistift-Verschlüsselungsverfahren

(Christine Stötzzel, April 2004; Updates B.+C. Esslinger, Juni 2005; Updates Minh Van Nguyen und B. Esslinger, November 2009 und Juni 2010)

*Edgar Allan Poe: A Few Words on Secret Writing, 1841*

Nur Wenige kann man glauben machen, dass es keine leichte Sache ist, eine Geheimschrift zu erdenken, die sich der Untersuchung widersetzt. Dennoch kann man rundheraus annehmen, dass menschlicher Scharfsinn keine Chiffre erdenken kann, die menschlicher Scharfsinn nicht lösen kann.

Das folgende Kapitel bietet einen recht vollständigen Überblick über Papier- und Bleistiftverfahren<sup>1</sup>. Unter diesem Begriff lassen sich alle Verfahren zusammenfassen, die Menschen von Hand anwenden können, um Nachrichten zu ver- und entschlüsseln. Besonders populär waren diese Verfahren für Geheimdienste (und sind es immer noch), da ein Schreibblock und ein Stift – im Gegensatz zu elektronischen Hilfsmitteln – vollkommen unverdächtig sind.

Die ersten Papier- und Bleistiftverfahren entstanden bereits vor rund 3000 Jahren, aber auch während des vergangenen Jahrhunderts kamen noch zahlreiche neue Methoden hinzu. Bei allen Papier- und Bleistiftverfahren handelt es sich um symmetrische Verfahren. Selbst in den ältesten Verschlüsselungsmethoden steckten schon die grundsätzlichen Konstruktionsprinzipien wie Transposition, Substitution, Blockbildung und deren Kombination. Daher lohnt es sich vor allem aus didaktischen Gesichtspunkten, diese „alten“ Verfahren genauer zu betrachten.

Erfolgreiche bzw. verbreiteter eingesetzte Verfahren mussten die gleichen Merkmale erfüllen wie moderne Verfahren:

- Vollständige Beschreibung, klare Regeln, ja fast Standardisierung (inkl. der Sonderfälle, dem Padding, etc.).
- Gute Balance zwischen Sicherheit und Benutzbarkeit (denn zu kompliziert zu bedienende Verfahren waren fehlerträchtig oder unangemessen langsam).

<sup>1</sup>In den Fußnoten zu diesem Kapitel wird jeweils beschrieben, wie man diese Verfahren mit CrypTool 1 ausführen kann. Außerdem enthält das letzte Unterkapitel (2.5) zu einigen Verfahren Beispielcode in dem Computer-Algebra-System Sage.

## 2.1 Transpositionsverfahren

Bei der Verschlüsselung durch Transposition bleiben die ursprünglichen Zeichen der Nachricht erhalten, nur ihre Anordnung wird geändert (Transposition = Vertauschung)<sup>2</sup>.

### 2.1.1 Einführungs-Beispiele unterschiedlicher Transpositionsverfahren

- **Gartenzaun**<sup>3</sup> [Singh2001]: Die Buchstaben des Klartextes werden abwechselnd in zwei (oder mehr) Zeilen geschrieben, so dass ein Zickzack-Muster entsteht. Dann werden die Zeichen zeilenweise nacheinander gelesen.  
Dieses Verfahren ist eher eine Kinderverschlüsselung.

Klartext<sup>4</sup>: ein Beispiel zur Transposition

```
i b i p e z r r n p s t o  
e n e s i l u t a s o i i n
```

Tabelle 2.1: Gartenzaun-Verschlüsselung

Geheimtext<sup>5</sup>: IBIPE ZRRNP STOEN ESILU TASOI IN

- **Skytale von Sparta**<sup>6</sup> [Singh2001]: Dieses Verfahren wurde wahrscheinlich das erste Mal um 600 v.Chr. benutzt und es wurde von dem griechischen Schriftsteller und Philosophen Plutarch (50-120 v.Chr.) zuerst beschrieben.  
Um einen Holzstab wird ein Streifen Papier o.ä. gewickelt. Dann wird darauf zeilenweise der Klartext geschrieben. Wickelt man den Streifen ab, hält man den Geheimtext in den Händen.
- **Schablone** [Goebel2003]: Sender und Empfänger benutzen die gleiche Schablone. In deren Löcher werden zeilenweise die Klartextzeichen geschrieben, die dann spaltenweise ausgelesen werden. Bleibt Klartext übrig, wird der Vorgang wiederholt, unter Umständen mit einer anderen Ausrichtung der Schablone<sup>7</sup>.

<sup>2</sup>Ein anderer Begriff für eine Transposition ist Permutation.

<sup>3</sup>Dieses Verfahren kann direkt in CrypTool mit dem Menüpunkt **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Skytale / Gartenzaun** abgebildet werden. Man kann diese Methode auch simulieren über den Menüeintrag **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Permutation**: für einen Gartenzaun mit 2 Zeilen gibt man als Schlüssel „B,A“ ein und lässt ansonsten die Standardeinstellungen (nur 1 Permutation, in der man zeilenweise ein- und spaltenweise ausliest). Mit dem Schlüssel „A,B“ würde man man das Zick-zack-Muster unten so beginnen, dass der 1. Buchstabe in der ersten statt in der 2. Zeile steht.

<sup>4</sup>Wenn das Alphabet nur die 26 Buchstaben verwendet, schreiben wir den Klartext in Kleinbuchstaben und den Geheimtext in Großbuchstaben.

<sup>5</sup>Die Buchstaben des Klartextes sind hier – wie historisch üblich – in 5-er Blöcken gruppiert. Man könnte o.E.d.A. auch eine andere (konstante) Blocklänge oder gar keine Trennung durch Leerzeichen wählen.

<sup>6</sup>Dieses Verfahren kann direkt in CrypTool mit dem Menüpunkt **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Skytale / Gartenzaun** abgebildet werden. Da dieses Verschlüsselungsverfahren ein Spezialfall der einfachen Spaltentransposition ist, kann man es in CrypTool simulieren über den Menüeintrag **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Permutation**: Für die Skytale braucht man in der Dialogbox nur die erste Permutation. Darin gibt man bei z.B. 4 Kanten als Schlüssel „1,2,3,4“ ein. Dies wäre so, als würde man den Text in 4-er Blöcken waagrecht in eine Tabelle schreiben und senkrecht auslesen. Weil der Schlüssel aufsteigend geordnet ist, bezeichnet man die Skytale auch als identische Permutation. Weil das Schreiben und Auslesen nur einmal durchgeführt wird, als einfache (und nicht als doppelte) Permutation.

<sup>7</sup>Dieses Verfahren kann man nicht durch eine einfache Spaltentransposition darstellen.

- **Fleißner-Schablone** [Savard1999]: Die Fleißner-Schablone wurde im Ersten Weltkrieg von deutschen Soldaten benutzt<sup>8</sup>. Ein quadratisches Gitter dient als Schablone, wobei ein Viertel der Felder Löcher hat. Der erste Teil des Klartextes wird zeilenweise durch die Löcher auf ein Blatt Papier geschrieben, dann wird die Schablone um 90 Grad gedreht, und der zweite Teil des Textes wird auf das Papier geschrieben, usw. Die Kunst besteht in der richtigen Wahl der Löcher: Kein Feld auf dem Papier darf frei bleiben, es darf aber auch keines doppelt beschriftet werden. Der Geheimtext wird zeilenweise ausgelesen.

In die Beispiel-Fleißner-Schablone in der folgenden Tabelle können 4 Mal je 16 Zeichen des Klartextes auf ein Blatt geschrieben werden:

O	-	-	-	-	O	-	-
-	-	-	O	O	-	-	O
-	-	-	O	-	-	O	-
-	-	O	-	-	-	-	-
-	-	-	-	O	-	-	-
O	-	O	-	-	-	O	-
-	O	-	-	-	-	-	O
-	-	-	O	O	-	-	-

Tabelle 2.2: 8x8-Fleißner-Schablone

### 2.1.2 Spalten- und Zeilentranspositionsverfahren<sup>9</sup>

- **Einfache Spaltentransposition** [Savard1999]: Zunächst wird ein Schlüsselwort bestimmt, das über die Spalten eines Gitters geschrieben wird. Dann schreibt man den zu verschlüsselnden Text zeilenweise in dieses Gitter. Die Spalten werden entsprechend des Auftretens der Buchstaben des Schlüsselwortes im Alphabet durchnummeriert. In dieser Reihenfolge werden nun auch die Spalten ausgelesen und so der Geheimtext gebildet<sup>10</sup>.

Klartext: ein beispiel zur transposition

Transpositionsschlüssel: K=2; E=1; Y=3.

Geheimtext: IEPLR APIOE BSEUR SSINI IZTNO TN

- **AMSCO** [ACA2002]: Die Klartextzeichen werden abwechselnd in Einer- und Zweiergruppen in ein Gitter geschrieben. Dann erfolgt eine Vertauschung der Spalten, anschließend das Auslesen.
- **Doppelte Spaltentransposition / „Doppelwürfel“** [Savard1999]: Die doppelte Spaltentransposition wurde häufig im Zweiten Weltkrieg und zu Zeiten des Kalten Krieges angewendet. Dabei werden zwei Spaltentranspositionen nacheinander durchgeführt, für die zweite Transposition wird ein neuer Schlüssel benutzt<sup>11</sup>.

<sup>8</sup>Erfunden wurde die Fleißner-Schablone bereits 1881 von Eduard Fleißner von Wostrowitz.

Eine gute Visualisierung findet sich unter [www.turning-grille.com](http://www.turning-grille.com).

<sup>9</sup>Die meisten der folgenden Verfahren können in CrypTool mit dem Menüpunkt **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Permutation** abgebildet werden.

<sup>10</sup>Darstellung mit CrypTool: Eingabe eines Schlüssels für die 1. Permutation, zeilenweise einlesen, spaltenweise permutieren und auslesen.

<sup>11</sup>Darstellung mit CrypTool: Eingabe eines Schlüssels für die 1. Permutation, zeilenweise einlesen, spaltenweise

K	E	Y
e	i	n
b	e	i
s	p	i
e	l	z
u	r	t
r	a	n
s	p	o
s	i	t
i	o	n

Tabelle 2.3: Einfache Spaltentransposition

- **Spaltentransposition, General Luigi Sacco** [Savard1999]: Die Spalten eines Gitters werden den Buchstaben des Schlüsselwortes entsprechend nummeriert. Der Klartext wird dann zeilenweise eingetragen, in der ersten Zeile bis zur Spalte mit der Nummer 1, in der zweiten Zeile bis zur Spalte mit der Nummer 2 usw. Das Auslesen erfolgt wiederum spaltenweise.

Klartext: ein beispiel zur transposition

G	E	N	E	R	A	L
4	2	6	3	7	1	5
e	i	n	b	e	i	
s	p					
i	e	l	z			
u						
r	t	r	a	n	s	p
o	s	i				
t	i	o	n			

Tabelle 2.4: Spaltentransposition nach General Luigi Sacco

Geheimtext: ESIUR OTIPE TSINL RIOBZ ANENI SP

- **Spaltentransposition, Französische Armee im Ersten Weltkrieg** [Savard1999]: Nach Durchführung einer Spaltentransposition werden diagonale Reihen ausgelesen.
- **Zeilentransposition** [Savard1999]: Der Klartext wird in gleich lange Blöcke zerlegt, was auch mit Hilfe eines Gitters erfolgen kann. Dann wird die Reihenfolge der Buchstaben bzw. der Spalten vertauscht. Da das Auslesen zeilenweise erfolgt, wird nur jeweils innerhalb der Blöcke permutiert<sup>12</sup>.

---

permutieren und auslesen. Eingabe eines (neuen) Schlüssels für die 2. Permutation, Ergebnis der 1. Permutation zeilenweise einlesen, spaltenweise permutieren und auslesen.

<sup>12</sup>Darstellung mit CrypTool: Eingabe eines Schlüssels für die 1. Permutation, zeilenweise einlesen, spaltenweise permutieren und zeilenweise auslesen.

### 2.1.3 Weitere Transpositionsverfahren

- **Geometrische Figuren** [Goebel2003]: Einem bestimmten Muster folgend wird der Klartext in ein Gitter geschrieben (Schnecke, Rösselsprung o.ä.), einem zweiten Muster folgend wird der Geheimtext ausgelesen.
- **Union Route Cipher** [Goebel2003]: Die Union Route Cipher hat ihren Ursprung im Amerikanischen Bürgerkrieg. Nicht die Buchstaben einer Nachricht werden umsortiert, sondern die Wörter. Für besonders prägnante Namen und Bezeichnungen gibt es Codewörter, die zusammen mit den Routen in einem Codebuch festgehalten werden. Eine Route bestimmt die Größe des Gitters, in das der Klartext eingetragen wird und das Muster, nach dem der Geheimtext ausgelesen wird. Zusätzlich gibt es eine Anzahl von Füllwörtern.
- **Nihilist-Transposition** [ACA2002]: Der Klartext wird in eine quadratische Matrix eingetragen, an deren Seiten jeweils der gleiche Schlüssel steht. Anhand dieses Schlüssels werden sowohl die Zeilen als auch die Spalten alphabetisch geordnet und der Inhalt der Matrix dementsprechend umsortiert. Der Geheimtext wird zeilenweise ausgelesen.

Klartext: ein beispiel zur transposition

	K	A	T	Z	E		A	E	K	T	Z
K	e	i	n	b	e	A	s	e	i	p	i
A	i	s	p	i	e	E	s	i	o	i	t
T	l	z	u	r	t	K	i	e	e	n	b
Z	r	a	n	s	p	T	z	t	l	u	r
E	o	s	i	t	i	Z	a	p	r	n	s

Tabelle 2.5: Nihilist-Transposition<sup>13</sup>

Geheimtext: SEIPI SIOIT IEENB ZTLUR APRNS

- **Cadenus** [ACA2002]: Hierbei handelt es sich um eine Spaltentransposition, die zwei Schlüsselworte benutzt.

Das erste Schlüsselwort wird benutzt, um die Spalten zu vertauschen.

Das zweite Schlüsselwort wird benutzt, um das Startzeichen jeder Spalte festzulegen: dieses zweite Schlüsselwort ist eine beliebige Permutation des benutzten Alphabets. Diese schreibt man links vor die erste Spalte. Jede Spalte wird dann vertikal so verschoben (wrap-around), dass sie mit dem Buchstaben beginnt, der in derjenigen Zeile, wo der Schlüsselbuchstabe des ersten Schlüsselwortes in dem zweiten Schlüsselwort zu finden ist. Der Geheimtext wird zeilenweise ausgelesen.

Siehe Tabelle 2.6.

Klartext: ein laengeres beispiel zur transposition mit cadenus

Geheimtext: PRNIB OOSTI ENAUT NRDIS UASNG IEEME ECSPE ILSIR EZALT N

<sup>13</sup>Der linke Block ist das Resultat nach dem Einlesen. Der rechte Block ist das Resultat nach dem Vertauschen von Zeilen und Spalten.

<sup>14</sup>In dem zweiten Dreierblock sind diejenigen Zeichen fett, die nach der Anwendung des zweiten Schlüsselwortes oben im dritten Dreierblock stehen.



	K	<b>E</b>	Y	<b>E</b>	K	Y	<b>E</b>	K	Y
A	e	i	n	i	e	n	<b>p</b>	r	n
D	l	a	e	a	l	e	i	b	o
X	n	g	e	g	n	e	o	s	t
K	r	e	s	e	<b>r</b>	s	i	e	n
C	b	e	i	e	b	i	a	u	t
W	s	p	i	p	s	i	n	r	d
N	e	l	z	l	e	z	i	s	u
S	u	r	t	r	u	t	a	s	n
Y	r	a	n	a	r	<b>n</b>	g	i	e
<b>E</b>	s	<b>p</b>	o	<b>p</b>	s	o	e	m	e
D	s	i	t	i	s	t	e	c	s
T	i	o	n	o	i	n	p	e	i
U	m	i	t	i	m	t	l	s	i
B	c	a	d	a	c	d	r	e	z
R	e	n	u	n	e	u	a	l	t
G	s	-	-	-	s	-	-	n	-

Tabelle 2.6: Cadenus<sup>14</sup>

## 2.2 Substitutionsverfahren

### 2.2.1 Monoalphabetische Substitutionsverfahren

Monoalphabetische Substitutionsverfahren ordnen jedem Klartextzeichen ein Geheimtextzeichen fest zu, d.h. diese Zuordnung ist während des ganzen Verschlüsselungsprozesses dieselbe.

- **Allgemeine monoalphabetische Substitution / Zufällige Buchstabenzuordnung**<sup>15</sup> [Singh2001]: Die Substitution erfolgt aufgrund einer festgelegten Zuordnung der Einzelbuchstaben.
- **Atbash**<sup>16</sup> [Singh2001]: Der erste Buchstabe des Alphabets wird durch den letzten Buchstaben des Alphabets ersetzt, der zweite durch den vorletzten, usw.
- **Verschiebechiffre, z.B. Caesar**<sup>17</sup> [Singh2001]: Klartext- und Geheimtextalphabet werden um eine bestimmte Anzahl von Zeichen gegeneinander verschoben.

Klartext:

bei der caesarchiffre wird um drei stellen verschoben

Geheimtext:

EHL GHU FDHVDUFKLIIUH ZLUG XP GUHL VWHOQH YHUVFKREHQ

- **Affine Chiffre**: Dies ist eine Verallgemeinerung der Verschiebechiffre. Jeder Klartextbuchstabe wird erst durch einen anderen ersetzt und das Ergebnis wird dann mit der Verschiebechiffre verschlüsselt. Die Bezeichnung „affine Chiffre“ kommt daher, dass man die Ver- und Entschlüsselungs-Funktion affine bzw. lineare Funktionen sind.
- **Substitution mit Symbolen, z.B. Freimaurerchiffre** [Singh2001]: Ein Buchstabe wird durch ein Symbol ersetzt.
- **Varianten**: Füller, absichtliche Fehler [Singh2001].
- **Nihilist-Substitution**<sup>18</sup> [ACA2002]: Das Alphabet wird in eine 5x5-Matrix eingetragen und bei der Verarbeitung wird jedem Klartextbuchstaben die aus Zeilen- und Spaltennummer gebildete Zahl zugeordnet. Dann wird ein Schlüsselwort gewählt und über eine zweite Tabelle geschrieben. In diese wird der Klartext zeilenweise eingetragen. Die Geheimtextzeichen sind die Summen aus den Zahlen des Klartextes und den Zahlen des Schlüsselwortes. Bei Zahlen zwischen 100 und 110 wird die führende „1“ ignoriert, so dass jeder Buchstabe durch eine zweistellige Zahl repräsentiert wird.

Siehe Tabelle 2.7.

Klartext: ein beispiel zur substitution

<sup>15</sup> Dieses Verfahren kann in CrypTool mit dem Menüpunkt **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Substitution / Atbash** abgebildet werden.

<sup>16</sup> Dieses Verfahren kann in CrypTool mit dem Menüpunkt **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Substitution / Atbash** abgebildet werden.

<sup>17</sup> In CrypTool kann man dieses Verfahren an drei verschiedenen Stellen im Menü finden:

- **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Caesar / ROT13**
- **Analyse \ Symmetrische Verschlüsselung (klassisch) \ Ciphertext only \ Caesar**
- **Einzelverfahren \ Visualisierung von Algorithmen \ Caesar**.

<sup>18</sup> Eine Animation zu diesem Nihilist-Verfahren findet sich in CrypTool unter dem Menüpunkt **Einzelverfahren \ Visualisierung von Algorithmen \ Nihilist**.

	1	2	3	4	5
1	S	C	H	L	U
2	E	A	B	D	F
3	G	I	K	M	N
4	O	P	Q	R	T
5	V	W	X	Y	Z

Matrix

K	E	Y
(33)	(21)	(54)
e	i	n
(54)	(53)	(89)
b	e	i
(56)	(42)	(86)
s	p	i
(44)	(63)	(86)
e	l	z
(54)	(35)	(109)
u	r	s
(48)	(65)	(65)
u	b	s
(48)	(44)	(65)
t	i	t
(78)	(53)	(99)
u	t	i
(48)	(66)	(86)
o	n	
(74)	(56)	

Tabelle

Tabelle 2.7: Nihilist-Substitution

Geheimtext: 54 53 89 56 42 86 44 63 86 54 35 09 48 65 65 48 44 65 78 53 99 48 66 86  
74 56

- **Codierung** [Singh2001]: Im Laufe der Geschichte wurden immer wieder Codebücher verwendet. In diesen Büchern wird jedem möglichen **Wort** eines Klartextes ein Codewort, ein Symbol oder eine Zahl zugeordnet. Voraussetzung für eine erfolgreiche geheime Kommunikation ist, dass Sender und Empfänger exakt das gleiche Codebuch besitzen und die Zuordnung der Codewörter zu den Klartextwörtern nicht offengelegt wird.
- **Nomenklatur** [Singh2001]: Eine Nomenklatur ist ein Verschlüsselungssystem, das auf einem Geheimtextalphabet basiert, mit dem ein Großteil der Nachricht chiffriert wird. Für besonders häufig auftretende oder geheim zu haltende Wörter existieren eine begrenzte Anzahl von Codewörtern.
- **Landkarten-Chiffre** [ThinkQuest1999]: Diese Methode stellt eine Kombination aus Substitution und Steganographie<sup>19</sup> dar. Klartextzeichen werden durch Symbole ersetzt, diese

<sup>19</sup>Statt eine Nachricht zu verschlüsseln, versucht man bei der reinen Steganographie, die Existenz der Nachricht zu verbergen.

werden nach bestimmten Regeln in Landkarten angeordnet.

- **Straddling Checkerboard** [Goebel2003]: Eine 3x10-Matrix wird mit den Buchstaben des Alphabets und zwei beliebigen Sonderzeichen oder Zahlen gefüllt, indem zunächst die voneinander verschiedenen Zeichen eines Schlüsselwortes und anschließend die restlichen Buchstaben des Alphabets eingefügt werden. Die Spalten der Matrix werden mit den Ziffern 0 bis 9, die zweite und dritte Zeile der Matrix mit den Ziffern 1 und 2 nummeriert. Jedes Zeichen des Geheimtextes wird durch die entsprechende Ziffer bzw. das entsprechende Ziffern paar ersetzt. Da die 1 und die 2 die ersten Ziffern der möglichen Ziffern kombinationen sind, werden sie nicht als einzelne Ziffern verwendet.

Siehe Tabelle 2.8.

Klartext: substitution bedeutet ersetzung

	0	1	2	3	4	5	6	7	8	9
	S	-	-	C	H	L	U	E	A	B
1	D	F	G	I	J	K	M	N	O	P
2	Q	R	T	V	W	X	Y	Z	.	/

Tabelle 2.8: Straddling Checkerboard mit Passwort „Schluessel“

Geheimtext: 06902 21322 23221 31817 97107 62272 27210 72227 61712

Auffällig ist die Häufigkeit der Ziffern 1 und 2, dies wird jedoch durch die folgende Variante behoben.

- **Straddling Checkerboard, Variante** [Goebel2003]: Diese Form des Straddling Checkerboards wurde von sowjetischen Spionen im Zweiten Weltkrieg entwickelt. Angeblich haben auch Ernesto (Ché) Guevara und Fidel Castro diese Chiffre zur geheimen Kommunikation benutzt. Das Alphabet wird in ein Gitter eingetragen (Spaltenanzahl = Länge des Schlüsselwortes), und es werden zwei beliebige Ziffern als „reserviert“ festgelegt, die später die zweite und dritte Zeile einer 3x10-Matrix bezeichnen (in unserem Bsp. 3 und 7). Nun wird das Gitter mit dem erzeugten Alphabet spaltenweise durchlaufen und Buchstaben zeilenweise in die Matrix übertragen: Die acht häufigsten Buchstaben (ENIR-SATD für die deutsche Sprache) bekommen zur schnelleren Chiffrierung die Ziffern 0 bis 9 zugewiesen, dabei werden die reservierten Ziffern nicht vergeben. Die übrigen Buchstaben werden der Reihe nach in die Matrix eingetragen. Gegebenenfalls wird als zweite Stufe der Verschlüsselung zum Geheimtext noch eine beliebige Ziffernfolge addiert.

Siehe Tabelle 2.9.

Klartext: substitution bedeutet ersetzung

Geheimtext: 07434 05957 45976 63484 87458 58208 53674 675

- **Ché Guevara**; Einen Spezialfall dieser Variante benutzte Ché Guevara (mit einem zusätzlicher Substitutionsschritt und einem leicht modifizierten Checkerboard):

- \* Die sieben häufigsten Buchstaben im Spanischen werden auf die erste Zeile verteilt.

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

Matrix		0	1	2	3	4	5	6	7	8	9
		<b>S</b>	<b>A</b>	<b>R</b>	-	<b>D</b>	<b>T</b>	<b>N</b>	-	<b>E</b>	<b>I</b>
	3	J	Q	Y	C	B	K	Z	H	M	.
	7	L	F	V	/	U	G	O	W	P	X

Tabelle 2.9: Variante des Straddling Checkerboards

- \* Es werden vier statt drei Zeilen benutzt.
- \* Damit konnte man  $10 * 4 - 4 = 36$  verschiedene Zeichen verschlüsseln.

- **Tri-Digital** [ACA2002]: Aus einem Schlüsselwort der Länge 10 wird ein numerischer Schlüssel gebildet, indem die Buchstaben entsprechend ihres Auftretens im Alphabet durchnummeriert werden. Dieser Schlüssel wird über ein Gitter mit zehn Spalten geschrieben. In dieses Gitter wird unter Verwendung eines Schlüsselwortes zeilenweise das Alphabet eingetragen, wobei die letzte Spalte frei bleibt. Die Klartextzeichen werden durch die Zahl über der entsprechenden Spalte substituiert, die Zahl über der freien Spalte dient als Trennzeichen zwischen den einzelnen Wörtern.
- **Baconian Cipher** [ACA2002]: Jedem Buchstaben des Alphabets und 6 Zahlen oder Sonderzeichen wird ein fünfstelliger Binärcode zugeordnet (zum Beispiel 00000 = A, 00001 = B, usw.). Die Zeichen der Nachricht werden entsprechend ersetzt. Nun benutzt man eine zweite, unverdächtige Nachricht, um darin den Geheimtext zu verbergen. Dies kann zum Beispiel durch Klein- und Großschreibung oder kursiv gesetzte Buchstaben geschehen: man schreibt z.B. alle Buchstaben in der unverdächtigen Nachricht groß, die unter einer „1“ stehen.

Siehe Tabelle 2.10.

Nachricht	H	I	L	F	E
Geheimtext	00111	01000	01011	00101	00100
Unverdächtige Nachricht	esist	warmu	nddie	sonne	scheint
Baconian Cipher	esIST	wArmU	nDdIE	soNnE	scHeint

Tabelle 2.10: Baconian Cipher

### 2.2.2 Homophone Substitutionsverfahren

Homophone Verfahren stellen eine Sonderform der monoalphabetischen Substitution dar. Jedem Klartextzeichen werden mehrere Geheimtextzeichen zugeordnet.

- **Homophone monoalphabetische Substitution**<sup>20</sup> [Singh2001]: Um die typische Häufigkeitsverteilung der Buchstaben einer natürlichen Sprache zu verschleiern, werden einem Klartextbuchstaben mehrere Geheimtextzeichen fest zugeordnet. Die Anzahl der zugeordneten Zeichen richtet sich gewöhnlich nach der Häufigkeit des zu verschlüsselnden Buchstabens.
- **Beale-Chiffre** [Singh2001]: Die Beale-Chiffre ist eine Buchchiffre, bei der die Wörter eines Schlüsseltextes durchnummeriert werden. Diese Zahlen ersetzen die Buchstaben des Klartextes durch die Anfangsbuchstaben der Wörter.
- **Grandpré Cipher** [Savard1999]: Eine 10x10-Matrix (auch andere Größen sind möglich) wird mit zehn Wörter mit je zehn Buchstaben gefüllt, so dass die Anfangsbuchstaben ein elftes Wort ergeben. Da die Spalten und Zeilen mit den Ziffern 0 bis 9 durchnummeriert werden, lässt sich jeder Buchstabe durch ein Ziffernpaar darstellen. Es ist offensichtlich, dass bei einhundert Feldern die meisten Buchstaben durch mehrere Ziffernpaare ersetzt werden können. Wichtig ist, dass die zehn Wörter möglichst alle Buchstaben des Alphabets enthalten.
- **Buchchiffre**: Die Wörter eines Klartextes werden durch Zahlentripel der Form „Seite-Zeile-Position“ ersetzt. Diese Methode setzt eine genaue Absprache des verwendeten Buches voraus, so muss es sich insbesondere um die gleiche Ausgabe handeln (Layout, Fehlerkorrekturen, etc.).

### 2.2.3 Polygraphische Substitutionsverfahren

Bei der polygraphische Substitution werden keine einzelne Buchstaben ersetzt, sondern Buchstabengruppen. Dabei kann es sich um Digramme, Trigramme, Silben, etc. handeln.

- **Große Chiffre** [Singh2001]: Die Große Chiffre wurde von Ludwig XIV. verwendet und erst kurz vor Beginn des 20. Jahrhunderts entschlüsselt. Die Kryptogramme enthielten 587 verschieden Zahlen, jede Zahl repräsentierte eine Silbe. Die Erfinder dieser Chiffre (Rossignol, Vater und Sohn) hatten zusätzliche Fallen eingebaut, um die Sicherheit der Methode zu erhöhen. Eine Zahl konnte beispielsweise die vorangehende löschen oder ihr eine andere Bedeutung zuweisen.
- **Playfair**<sup>21</sup> [Singh2001]: Eine 5x5-Matrix wird mit dem Alphabet gefüllt, z.B. erst mit den verschiedenen Zeichen eines Schlüsselwortes, dann mit den restlichen Buchstaben des Alphabets. Der Klartext wird in Digramme unterteilt, die nach den folgenden Regeln verschlüsselt werden:
  1. Befinden sich die Buchstaben in der selben Spalte, werden sie durch die Buchstaben ersetzt, die direkt darunter stehen.
  2. Befinden sich die Buchstaben in der selben Zeile, nimmt man jeweils den Buchstaben rechts vom Klartextzeichen.
  3. Befinden sich die Buchstaben in unterschiedlichen Spalten und Zeilen, nimmt man jeweils den Buchstaben, der zwar in der selben Zeile, aber in der Spalte des anderen Buchstabens steht.

<sup>20</sup>In CrypTool kann man dieses Verfahren über den Menüeintrag **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Homophone** aufrufen.

<sup>21</sup>In CrypTool kann man dieses Verfahren über den Menüeintrag **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Playfair** aufrufen.

4. Für Doppelbuchstaben (falls sie in einem Digramm vorkommen) gelten Sonderregelungen, wie zum Beispiel die Trennung durch einen Füller.

Siehe Tabelle 2.11.

Klartext: buchstaben werden paarweise verschluesxst

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

Tabelle 2.11: 5x5-Playfair-Matrix

Geheimtext: FHHLU OBDFG VAYMF GWIDP VAGCG SDOCH LUSFH VEGUR

- **Playfair für Trigramme** [Savard1999]: Zunächst füllt man eine 5x5-Matrix mit dem Alphabet und teilt den Klartext in Trigramme auf. Für die Verschlüsselung gelten folgende Regeln:
  1. Drei gleiche Zeichen werden durch drei gleiche Zeichen ersetzt, es wird der Buchstabe verwendet, der rechts unter dem ursprünglichen Buchstaben steht (Beispiel anhand von Tabelle 11: BBB  $\Rightarrow$  MMM).
  2. Bei zwei gleichen Buchstaben in einem Trigramm gelten die Regeln der Original-Playfair-Verschlüsselung.
  3. Bei drei unterschiedlichen Buchstaben kommen relativ komplizierte Regeln zur Anwendung, mehr dazu unter [Savard1999].
- **Ersetzung von Digrammen durch Symbole** [Savard1999]: Giovanni Battista della Porta, 15. Jahrhundert. Er benutzte eine 20x20-Matrix, in die er für jede mögliche Buchstabenkombination (das verwendete Alphabet bestand aus nur zwanzig Zeichen) ein Symbol eintrug.
- **Four Square Cipher** [Savard1999]: Diese Methode ähnelt Playfair, denn es handelt sich um ein Koordinatensystem, dessen vier Quadranten jeweils mit dem Alphabet gefüllt werden, wobei die Anordnung des Alphabets von Quadrant zu Quadrant unterschiedlich sein kann. Um eine Botschaft zu verschlüsseln, geht man wie folgt vor: Man sucht den ersten Klartextbuchstaben im ersten Quadranten und den zweiten Klartextbuchstaben im dritten Quadranten. Denkt man sich ein Rechteck mit den beiden Klartextbuchstaben als gegenüberliegende Eckpunkte, erhält man im zweiten und vierten Quadranten die zugehörigen Geheimtextzeichen.

Siehe Tabelle 2.12.

Klartext: buchstaben werden paarweise verschluesst

Geheimtext: YNKHM XFVCI LAIPC IGRWP LACXC BVIRG MKUUR XVKT

- **Two Square Cipher** [Savard1999]: Die Vorgehensweise gleicht der der Four Square Cipher, allerdings enthält die Matrix nur zwei Quadranten. Befinden sich die beiden zu

d	w	x	y	m	E	P	T	O	L
r	q	e	k	i	C	V	I	Q	Z
u	v	h	p	s	R	M	A	G	U
a	l	<b>b</b>	z	n	F	W	<b>Y</b>	H	S
g	c	o	f	t	B	N	D	X	K
Q	T	B	L	E	v	q	i	p	g
Z	H	<b>N</b>	D	X	s	t	<b>u</b>	o	h
P	M	I	Y	C	n	r	d	x	y
V	S	K	W	O	b	l	w	m	f
U	A	F	R	G	c	z	k	a	e

Tabelle 2.12: Four Square Cipher

ersetzenden Buchstaben in der gleichen Reihe, werden sie nur vertauscht. Andernfalls werden die beiden Klartextzeichen als gegenüberliegende Eckpunkte eines Rechtecks betrachtet und durch die anderen Eckpunkte ersetzt. Die Anordnung der beiden Quadranten ist horizontal und vertikal möglich.

- **Tri Square Cipher** [ACA2002]: Drei Quadranten werden jeweils mit dem Alphabet gefüllt. Der erste Klartextbuchstabe wird im ersten Quadranten gesucht und kann mit jedem Zeichen der selben Spalte verschlüsselt werden. Der zweite Klartextbuchstabe wird im zweiten Quadranten (diagonal gegenüberliegend) gesucht und kann mit jedem Buchstaben derselben Zeile verschlüsselt werden. Zwischen diese Geheimtextzeichen wird der Buchstabe des Schnittpunktes gesetzt.
- **Dockyard Cipher/Werftschlüssel** [Savard1999]: Angewendet von der Deutschen Marine im Zweiten Weltkrieg.

## 2.2.4 Polyalphabetische Substitutionsverfahren

Bei der polyalphabetischen Substitution ist die Zuordnung Klartext-/Geheimtextzeichen nicht fest, sondern variabel (meist abhängig vom Schlüssel).

- **Vigenère**<sup>22</sup> [Singh2001]: Entsprechend den Zeichen eines Schlüsselwortes wird jedes Klartextzeichen mit einem anderen Geheimtextalphabet verschlüsselt (als Hilfsmittel dient das sog. Vigenère-Tableau). Ist der Klartext länger als der Schlüssel, wird dieser wiederholt. Siehe Tabelle 2.13.
  - **Unterbrochener Schlüssel**: Der Schlüssel wird nicht fortlaufend wiederholt, sondern beginnt mit jedem neuen Klartextwort von vorne.
  - **Autokey-Variante** [Savard1999]: Nachdem der vereinbarte Schlüssel abgearbeitet wurde, geht man dazu über, die Zeichen der Nachricht als Schlüssel zu benutzen. Siehe Tabelle 2.14.

<sup>22</sup>In CrypTool kann man dieses Verfahren über den Menüeintrag **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Vigenère** aufrufen.



Klartext:	das	alphabet	wechselt	staendig
Schlüssel:	KEY	KEYKEYKE	YKEYKEYK	EYKEYKEY
Geheimtext:	NEQ	KPNREZOX	UOGFCIJD	WRKILNME

-	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Tabelle 2.13: Vigenère-Tableau

Klartext:	das	alphabet	wechselt	staendig
Schlüssel:	KEY	DASALPHA	BETWECHS	ELTSTAEN
Geheimtext:	NEQ	DLHHLQLA	XIVDWGSL	WETWGMT

Tabelle 2.14: Autokey-Variante

- **Progressive-Key-Variante** [Savard1999]: Der Schlüssel ändert sich im Laufe der Chiffrierung, indem er das Alphabet durchläuft. So wird aus KEY LFZ.
- **Gronsfeld** [Savard1999]: Vigenère-Variante, die einen Zahlenschlüssel verwendet.
- **Beaufort** [Savard1999]: Vigenère-Variante, keine Verschlüsselung durch Addition, sondern durch Subtraktion. Auch mit rückwärts geschriebenem Alphabet.
- **Porta** [ACA2002]: Vigenère-Variante, die nur 13 Alphabete verwendet. Das bedeutet, dass jeweils zwei Schlüsselbuchstaben dasselbe Geheimtextalphabet zugeordnet wird, und die erste und zweite Hälfte des Alphabets reziprok sind.
- **Slidefair** [ACA2002]: Kann als Vigenère-, Gronsfeld- oder Beaufort-Variante verwendet werden. Dieses Verfahren verschlüsselt Digramme. Den ersten Buchstaben sucht man im Klartextalphabet über dem Tableau, den zweiten in der Zeile, die dem Schlüsselbuchstaben entspricht. Diese beiden Punkte bilden gegenüberliegende Punkte eines gedachten Rechtecks, die verbleibenden Ecken bilden die Geheimtextzeichen.

#### • Superposition

- **Buchchiffre**: Addition eines Schlüsseltextes (z.B. aus einem Buch) zum Klartext.
- **Überlagerung mit einer Zahlenfolge**: Eine Möglichkeit sind mathematische Folgen wie die Fibonacci-Zahlen.

- **Phillips** [ACA2002]: Das Alphabet wird in eine 5x5-Matrix eingetragen. Dann werden 7 weitere Matrizen erzeugt, indem zunächst immer die erste, dann die zweite Zeile um eine Position nach unten verschoben wird. Der Klartext wird in Blöcke der Länge 5 unterteilt, die jeweils mit Hilfe einer Matrix verschlüsselt werden. Dazu wird jeweils der Buchstabe rechts unterhalb des Klartextzeichens verwendet.
- **Ragbaby** [ACA2002]: Zuerst wird ein Alphabet mit 24 Zeichen konstruiert. Die Zeichen des Klartextes werden durchnummeriert, wobei die Nummerierung der Zeichen des ersten Wortes mit 1 beginnt, die des zweiten Wortes mit 2 usw. Die Zahl 25 entspricht wieder der Zahl 1. Ein Buchstabe der Nachricht wird chiffriert, indem man im Alphabet entsprechend viele Buchstaben nach rechts geht.

Alphabet: SCHLUEABDFGIKMNOPQRTVWXZ

Klartext:	d a s	a l p h a b e t	w e c h s e l t	s t a e n d i g
Nummerierung:	1 2 3	2 3 4 5 6 7 8 9	3 4 5 6 7 8 9 10	4 5 6 7 8 9 10 11
Geheimtext:	F D L	D A V B K N M U	S F A D B M K E	U S K K X Q W W

Tabelle 2.15: Ragbaby

## 2.3 Kombination aus Substitution und Transposition

In der Geschichte der Kryptographie sind häufig Kombinationen der oben angeführten Verfahrensklassen anzutreffen. Diese haben sich - im Durchschnitt - als sicherer erwiesen als Verfahren, die nur auf einem der Prinzipien Transposition oder Substitution beruhen.

- **ADFG(V)X**<sup>23</sup> [Singh2001]: Die ADFG(V)X-Verschlüsselung wurde in Deutschland im ersten Weltkrieg entwickelt. Eine 5x5- oder 6x6-Matrix wird mit dem Alphabet gefüllt, die Spalten und Zeilen werden mit den Buchstaben ADFG(V)X versehen. Jedes Klartextzeichen wird durch das entsprechende Buchstabenpaar ersetzt. Abschließend wird auf dem so entstandenen Text eine (Zeilen-)Transposition durchgeführt.
- **Zerlegung von Buchstaben, auch Fractionation genannt** [Savard1999]: Sammelbegriff für die Verfahren, die erst ein Klartextzeichen durch mehrere Geheimtextzeichen verschlüsseln und auf diese Verschlüsselung dann eine Transposition anwenden, so dass die ursprünglich zusammengehörenden Geheimtextzeichen voneinander getrennt werden.
  - **Bifid/Polybius square/Checkerboard** [Goebel2003]: Bei der Grundform dieser Verschlüsselungsmethode wird eine 5x5-Matrix mit den Buchstaben des Alphabets gefüllt (siehe Playfair). Die Spalten und Zeilen dieser Matrix müssen durchnummeriert sein, damit jedes Zeichen des Klartextes durch ein Ziffernpaar (Zeile/Spalte) ersetzt werden kann. Die Zahlen können jede beliebige Permutation von (1,2,3,4,5,) sein. Dies ist ein „Schlüssel“ oder Konfigurations-Parameter dieses Verfahrens. Eine mögliche Variante besteht darin, Schlüsselwörter statt der Zahlen 1 bis 5 zu verwenden. Meist wird der Klartext vorher in Blöcke gleicher Länge zerlegt. Die Blocklänge

<sup>23</sup>In CrypTool kann man dieses Verfahren über den Menüeintrag **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ ADFGVX** aufrufen.

(hier 5) ist ein weiterer Konfigurations-Parameter diese Verfahrens. Um den Geheimtext zu erhalten, werden zunächst alle Zeilennummern, dann alle Spaltennummern eines Blocks ausgelesen. Anschließend werden die Ziffern paarweise in Buchstaben umgewandelt.

Siehe Tabelle 2.16.

	2	4	5	1	3
1	S	C	H	L	U
4	E	A	B	D	F
2	G	I	K	M	N
3	O	P	Q	R	T
5	V	W	X	Y	Z

Klartext:	kombi	natio	nenme	hrere	rverf	ahren
Zeilen:	23242	24323	24224	13434	35434	41342
Spalten:	52154	34342	32312	51212	12213	45123

Tabelle 2.16: Bifid

23242 52154 24323 34342 24224 32312 13434 51212 35434 12213 41342 45123

Geheimtext: NIKMW IOTFE IGFNS UFBSS QFDGU DPIYN

- **Trifid** [Savard1999]: 27 Zeichen (Alphabet + 1 Sonderzeichen) können durch Tripel aus den Ziffern 1 bis 3 repräsentiert werden. Die zu verschlüsselnde Botschaft wird in Blöcke der Länge 3 zerlegt und unter jeden Buchstaben wird das ihm entsprechende Zahlentripel geschrieben. Die Zahlen unter den Blöcken werden wiederum als Tripel zeilenweise ausgelesen und durch entsprechende Zeichen ersetzt.
- **Bazeries** [ACA2002]: Eine 5x5-Matrix wird spaltenweise mit dem Alphabet gefüllt, eine zweite Matrix wird zeilenweise mit dem Schlüssel (einer ausgeschriebene Zahl kleiner als 1.000.000) und den übrigen Buchstaben des Alphabets gefüllt. Der Text wird in beliebige Blöcke unterteilt, die Reihenfolge dieser Zeichen wird jeweils umgekehrt und zu jedem Zeichen entsprechend seiner Position in der ersten Matrix sein Gegenstück in der Schlüsselmatrix gesucht.

Siehe Tabelle 2.17.

Klartext: kombinationen mehrerer verfahren

Schlüsselwort: 900.004 (neunhunderttausendundvier)

- **Digrafid** [ACA2002]: Zur Substitution der Digramme wird die nachfolgende Matrix benutzt (der Einfachheit halber wird das Alphabet hier in seiner ursprünglichen Reihenfolge verwendet). Der erste Buchstabe eines Digramms wird im waagerechten Alphabet gesucht, notiert wird die Nummer der Spalte. Der zweite Buchstabe wird im senkrechten Alphabet gesucht, notiert wird die Nummer der Zeile. Zwischen diese beiden Ziffern wird die Ziffer des Schnittpunktes gesetzt. Die Tripel werden vertikal unter die Digramme, die in Dreierblöcken angeordnet sind, geschrieben. Dann werden die horizontal entstandenen dreistelligen Zahlen ausgelesen und in Buchstaben umgewandelt.

#### Bemerkung:

Da das Verfahren immer ganze Dreiergruppen benötigt, ist eine vollständige Verfahrens-

a	f	l	q	v	N	E	U	H	D
b	g	<b>m</b>	r	w	R	T	<b>A</b>	S	V
c	h	n	s	x	I	B	C	F	G
d	i	o	t	y	K	L	M	O	P
e	k	p	u	z	Q	W	X	Y	Z

kom	bi	nation	enm	ehr	ere	rverf	ahr	en
<b>mok</b>	ib	noitan	mne	rhe	ere	frevr	rha	ne
<b>AMW</b>	LR	CMLONC	ACQ	SBQ	QSQ	ESQDS	SBN	CQ

Tabelle 2.17: Bazerries

beschreibung zwischen Sender und Empfänger notwendig, die auch den Umgang mit Texten erläutert, die im letzten Block nur 1-5 Klartextbuchstaben enthalten. Vom Weglassen bis Padding mit zufällig oder fix vorher festgelegten Buchstaben ist alles möglich.

Siehe Tabelle 2.18.

1	2	3	4	5	6	7	8	9				
A	B	C	D	E	F	G	H	I	1	2	3	
J	K	L	M	N	O	P	Q	R	4	5	6	
S	T	U	V	W	X	Y	Z	.	7	8	9	
									A	J	S	1
									B	K	T	2
									C	L	U	3
									D	M	V	4
									E	N	W	5
									F	O	X	6
									G	P	Y	7
									H	Q	Z	8
									I	R	.	9

ko	mb	in	at	io	ne	nm	eh	re	re	rv	er	fa	hr	en
2	4	9	1	9	5	5	5	9	9	9	5	6	8	5
5	4	2	3	2	4	5	1	4	4	6	2	1	2	2
6	2	5	2	6	5	4	8	5	5	4	9	1	9	5
KI	NB	FN	SW	CM	KW	NR	ED	VN	.W	MT	NI	XN	AK	SW

Tabelle 2.18: Digrafid

- **Nicodemus** [ACA2002]: Zunächst wird eine einfache Spaltentransposition durchgeführt. Noch vor dem Auslesen erfolgt eine Vigenère-Verschlüsselung (die Buchstaben einer Spalte werden mit dem entsprechenden Zeichen des Schlüsselwortes chiffriert). Das Auslesen erfolgt in vertikalen Blöcken.

Siehe Tabelle 2.19.

Klartext: kombinationen mehrerer verfahren

Geheimtext: SMXRQ ULKYX KLGCC VIIIEI RBFPB CPPFL

K	E	Y	E	K	Y	E	K	Y
k	o	m	o	k	m	S	U	K
b	i	n	i	b	n	M	L	L
a	t	i	t	a	i	X	K	G
o	n	e	n	o	e	R	Y	C
n	m	e	m	n	e	Q	X	C
h	r	e	r	h	e	V	R	C
r	e	r	e	r	r	I	B	P
v	e	r	e	v	r	I	F	P
f	a	h	a	f	h	E	P	F
r	e	n	e	r	n	I	B	L

Tabelle 2.19: Nicodemus

## 2.4 Andere Verfahren

- **Nadelstich-Verschlüsselung** [Singh2001]: Dieses simple Verfahren wurde aus den unterschiedlichsten Gründen über viele Jahrhunderte hinweg praktiziert (eigentlich ein Verfahren der Steganografie). So markierten zum Beispiel im Viktorianischen Zeitalter kleine Löcher unter Buchstaben in Zeitungsartikeln die Zeichen des Klartextes, da das Versenden einer Zeitung sehr viel billiger war als das Porto für einen Brief.
- **Lochschablone**: Die Lochschablone ist auch unter der Bezeichnung Kardinal-Richelieu-Schlüssel bekannt. Eine Schablone wird über einen vorher vereinbarten Text gelegt und die Buchstaben, die sichtbar bleiben, bilden den Geheimtext.
- **Kartenspiele** [Savard1999]: Der Schlüssel wird mit Hilfe eines Kartenspiels und vorher festgelegter Regeln erzeugt. Alle im Folgenden genannten Verfahren sind als Papier- und Bleistiftverfahren ausgelegt, also ohne elektronische Hilfsmittel durchführbar. Ein Kartenspiel ist für Außenstehende unverdächtig, das Mischen der Karten bietet ein gewisses Maß an Zufälligkeit, die Werte der Karten lassen sich leicht in Buchstaben umwandeln und Transpositionen lassen sich ohne weitere Hilfsmittel (sei es schriftlich oder elektronisch) durchführen.

- **Solitaire (Bruce Schneier)**<sup>24</sup> [Schneier1999]: Sender und Empfänger der Botschaft müssen jeweils ein Kartenspiel besitzen, bei dem alle Karten in der gleichen Ausgangsanordnung im Stapel liegen. Mit den Karten wird ein Schlüsselstrom erzeugt, der ebenso viele Zeichen besitzen muss wie der zu verschlüsselnde Klartext.

Als Basis zur Erzeugung des Schlüssels wird ein gemischtes Bridge-Kartenspiel mit 54 Karten (As, 2 - 10, Bube, Dame, König in vier Farben + 2 Joker) benutzt. Der Kartenstapel wird dazu offen in die Hand genommen:

1. Der erste Joker wandert um eine Position nach hinten.
2. Der zweite Joker wandert um zwei Positionen nach hinten.
3. Die Karten über dem ersten Joker werden mit den Karten unter dem zweiten Joker vertauscht.
4. Der Wert der untersten Karte (1 bis 53; Kreuz, Karo, Herz, Pik; Joker = 53) wird notiert. Genau so viele Karten werden von oben abgezählt und mit den übrigen Karten vertauscht, wobei die unterste Karte des Stapels liegen bleibt.
5. Der Wert der obersten Karte wird notiert. Entsprechend viele Karten werden von oben abgezählt.
6. Der Wert der darauffolgenden Karte ist das erste Schlüsselzeichen, Kreuz und Herz = 1 bis 13, Karo und Pik = 14 bis 26. Handelt es sich um einen Joker, wird wieder bei Schritt 1 begonnen.

Diese 6 Schritte werden für jedes neue Schlüsselzeichen abgearbeitet. Dieser Vorgang dauert – von Hand – relativ lange (4 h für 300 Zeichen, je nach Übung) und erfordert hohe Konzentration.

Die Verschlüsselung erfolgt dann durch Addition mod 26. Sie geht im Vergleich zur Schlüsselstromerzeugung relativ rasch.

---

<sup>24</sup>In CrypTool kann man dieses Verfahren über den Menüeintrag **Ver-/Entschlüsseln \ Symmetrisch (klassisch) \ Solitaire** aufrufen.

- **Mirdek (Paul Crowley)** [Crowley2000]: Hierbei handelt es sich um ein relativ kompliziertes Verfahren, der Autor liefert aber anhand eines Beispiels eine sehr anschauliche Erklärung.
- **Playing Card Cipher (John Savard)** [Savard1999]: Dieses Verfahren verwendet ein bereits gemischtes Kartenspiel ohne Joker, wobei das Mischen gesondert geregelt ist. Ein Schlüssel wird erzeugt ,indem:

1. Der Stapel liegt verdeckt vor dem Anwender, der solange Karten aufdeckt und in einer Reihe ablegt, bis die Summe der Werte größer oder gleich 8 ist.
2. Ist die letzte Karte ein Bube, eine Dame oder ein König, wird der Wert dieser Karte notiert, andernfalls notiert man die Summe der Werte der ausgelegten Karten (eine Zahl zwischen 8 und 17). In einer zweiten Reihe wird nun die notierte Anzahl von Karten ausgelegt.
3. Die dann noch verbleibenden Karten werden reihenweise abgelegt und zwar in der ersten Reihe bis zur Position der niedrigsten Karte aus 2., in der zweiten Reihe bis zur Position der zweitniedrigsten Karte aus 2. usw. Sind 2 Karten gleich, ist rot niedriger zu bewerten als schwarz.
4. Die in 3. ausgeteilten Karten werden spaltenweise eingesammelt und auf einem Stapel offen abgelegt (Spaltentransposition). Dabei wird mit der Spalte unter der niedrigsten Karte begonnen.(aufgedeckt)
5. Die in 1. und 2. ausgeteilten Karten werden eingesammelt (die letzte Karte wird zuerst entfernt).
6. Der Stapel wird umgedreht, so dass die Karten verdeckt liegen. Im Anschluss werden die Schritte 1 bis 6 noch zweimal ausgeführt.

Um ein Schlüsselzeichen zu erzeugen, wird der Wert der ersten Karte, die nicht Bube, Dame oder König ist, notiert und die entsprechende Anzahl Karten abgezählt (auch der Wert dieser Karte muss zwischen 1 und 10 liegen). Die gleichen Schritte werden ausgehend von der letzten Karte angewendet. Die Werte der beiden ausgewählten Karten werden addiert und die letzte Stelle dieser Summe ist das Schlüsselzeichen.

- **VIC cipher** [Savard1999]: Dies ist ein extrem aufwändiges, aber verhältnismäßig sicheres Papier- und Bleistiftverfahren. Es wurde von sowjetischen Spionen eingesetzt. Unter anderem musste der Anwender aus einem Datum, einem Satzanfang und einer beliebigen fünfstelligen Zahl nach bestimmten Regeln zehn Pseudozufallszahlen erzeugen. Bei der Verschlüsselung findet unter anderem auch ein Straddling Checkerboard Verwendung. Eine genaue Beschreibung der Vorgehensweise findet sich unter [Savard1999].

## 2.5 Anhang: Beispiele mit Sage

Im Folgenden werden einige der klassischen Verfahren mit dem Open-Source Computer-Algebra-System **Sage**<sup>25</sup> implementiert. Der Code ist lauffähig und getestet mit Sage Version 4.2. Alle Verfahren sind im Kapitel 2 („Papier- und Bleistift-Verschlüsselungsverfahren“) erklärt.

Um die Beispielprogramme<sup>26</sup> leichter lesbar zu machen, hielten wir uns an die in der folgenden Grafik aufgeführten Bezeichnungen und Abläufe:

- Die Verschlüsselung besteht aus den Schritten codieren und verschlüsseln.
  - Beim Codieren werden im Klartext  $P$  die Zeichen auf Groß-/Kleinschreibung angepasst (je nach gegebenem Alphabet), und alle Nichtalphabetzeichen werden entfernt.
  - Durch die Verschlüsselung wird das Chifftrat  $C$  erzeugt.
- Die Entschlüsselung besteht ebenfalls aus den zwei Schritten entschlüsseln und decodieren.
  - Ein abschließender Decodier-Schritt ist nur nötig, wenn die benutzten Symbole im Alphabet nicht die ASCII-Zeichen sind.

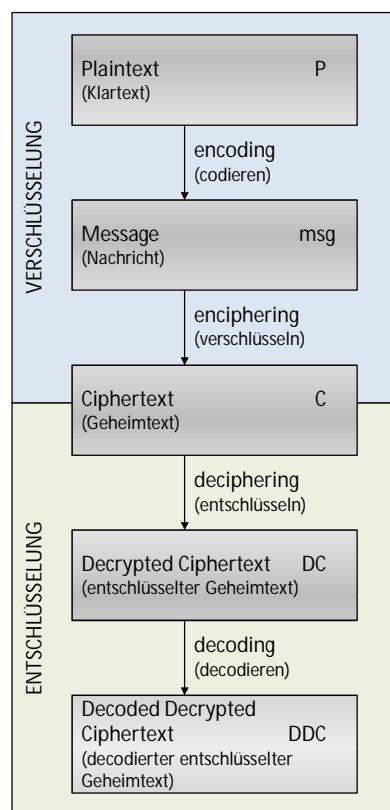


Abbildung 2.1: Namenskonventionen in den Sage-Programmbeispielen zu Verschlüsselungsverfahren

<sup>25</sup>Eine erste Einführung in das CAS Sage finden Sie im Anhang A.5.

<sup>26</sup>Weitere Programmbeispiele in Sage zu klassischen Kryptoverfahren finden sich z.B.:

- unter [http://www.sagemath.org/doc/constructions/linear\\_codes.html#classical-ciphers](http://www.sagemath.org/doc/constructions/linear_codes.html#classical-ciphers)
- in der Diplomarbeit von Minh Van Nguyen [Nguyen2009]



## 2.5.1 Transpositions-Chiffren

Transpositions-Chiffren sind implementiert in der Sage-Klasse

```
sage.crypto.classical.TranspositionCryptosystem
```

Bevor man mit einer Sage Transpositions-Chiffre arbeiten kann, muss man sie konstruieren: Dazu legt man das Alphabet fest, das sie Symbole (Zeichen) enthält, aus denen Klartext und Geheimtext bestehen können. Typischerweise besteht das Alphabet aus den normalen lateinischen Großbuchstaben. Dieses Alphabet legt man mit der folgenden Funktion fest

```
sage.monoids.string_monoid.AlphabeticStrings
```

Danach muss man die Blocklänge der Permutation festlegen, d.h. die Länge des Zeilenvektors der einfachen Spaltentransposition. Dieser Zeilenvektor ist der Schlüssel, mit dem die Buchstaben des Klartextes permutiert werden.

Im folgenden ersten Beispiel der Transpositions-Chiffren ist die Blocklänge 14, und der Schlüssel ist so gebaut, dass jeder Buchstabe im Klartext um zwei Zeichen nach rechts geschiftet wird (und wrap-around am Ende des Blocks). Das ist der Verschlüsselungsvorgang. Der Entschlüsselungsvorgang besteht im Shiften jedes Zeichens des Geheimtextes um  $14 - 2 = 12$  Zeichen nach links.

---

### Sage-Beispiel 2.1 Einfache Transposition durch Shiften (die Schlüssel sind explizit gegeben)

---

```
sage: # transposition cipher using a block length of 14
sage: T = TranspositionCryptosystem(AlphabeticStrings(), 14)
sage: # given plaintext
sage: P = "a b c d e f g h i j k l m n"
sage: # encryption key
sage: key = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1, 2]
sage:
sage: # encode plaintext (get rid of non-alphabet chars, convert lower-case to upper-case)
sage: msg = T.encoding(P)
sage: # encrypt plaintext by shifting to the left by 2 letters (do it in two steps)
sage: E = T(key)
sage: C = E(msg); C
CDEFGHIJKLMNAB
sage:
sage: # decrypt ciphertext by shifting to the left by 12 letters
sage: keyInv = [13, 14, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
sage: D = T(keyInv)
sage: D(C)
ABCDEFGHIJKLMN
sage:
sage: # Representation of key and inverse key as permutations
sage: E
(1,3,5,7,9,11,13)(2,4,6,8,10,12,14)
sage: D
(1,13,11,9,7,5,3)(2,14,12,10,8,6,4)
```

---

Das zweite Beispiel der Transpositions-Chiffren ist ebenfalls eine einfache shiftende Spaltentransposition. Aber hier ist der Programmcode ein wenig mehr automatisiert: Die Schlüssel werden anhand des Shift-Wertes generiert.

---

**Sage-Beispiel 2.2** Einfache Transposition durch Shiften (die Schlüssel werden mit „range“ konstruiert)

---

```
sage: # transposition cipher using a block length of 14, code more variable
sage: keylen = 14
sage: shift = 2
sage: A = AlphabeticStrings()
sage: T = TranspositionCryptosystem(A, keylen)
sage:
sage: # construct the plaintext string from the first 14 letters of the alphabet plus blanks
sage: # plaintext    = "A B C D E F G H I J K L M N"
sage: A.gens()
(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)
sage: P=' '
sage: for i in range(keylen): P=P + " " + str(A.gen(i))
.....:
sage: P
' A B C D E F G H I J K L M N'
sage:
sage: # encryption key
sage: # key = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1, 2]
sage: key = [(i+shift).mod(keylen) + 1 for i in range(keylen)]; key
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1, 2]
sage:
sage: # encode plaintext (get rid of non-alphabet chars)
sage: msg = T.encoding(P)
sage: # encrypt plaintext by shifting to the left by 2 letters (do it in one step)
sage: C = T.enciphering(key, msg); C
CDEFGHIJKLMNOPAB
sage:
sage: # decrypt ciphertext by shifting to the left by 12 letters
sage: # keyInv = [13, 14, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
sage: shiftInv=keylen-shift;
sage: keyInv = [(i+shiftInv).mod(keylen) + 1 for i in range(keylen)]; keyInv
[13, 14, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
sage: DC = T.deciphering(keyInv, C); DC
ABCDEFGHIJKLMN
sage:
sage: # decryption using the "deciphering method with key" instead of "enciphering with keyInv"
sage: # using the deciphering method requires to change the type of the variable key
sage: DC = T.deciphering(T(key).key(), C); DC
ABCDEFGHIJKLMN
sage:
sage: # representation of key and inverse key as permutations
sage: T(key)
(1,3,5,7,9,11,13)(2,4,6,8,10,12,14)
sage: T(key).key()
(1,3,5,7,9,11,13)(2,4,6,8,10,12,14)
sage: T(keyInv)
(1,13,11,9,7,5,3)(2,14,12,10,8,6,4)
```

---

Im dritten Beispiel der Transpositions-Chiffren wird eine beliebige Permutation als Schlüssel für die Ver- und Entschlüsselung gewählt, um die Zeichen in jedem Block zu verwürfeln (Blocklänge = Anzahl Spalten in der einfachen Spaltentransposition). Ist die Blocklänge  $n$ , dann ist der Schlüssel eine Permutation über  $n$  Zeichen. Das folgende Beispiel nutzt die Methode `random_key()` der Klasse `TranspositionCryptosystem`. Jeder Aufruf von `random_key()` erzeugt einen anderen Schlüssel. Deshalb werden bei Ihrem Aufruf voraussichtlich andere Ergebnisse (Schlüssel und Geheimtext) auftreten.

---

### Sage-Beispiel 2.3 Einfache Spalten-Transposition mit zufällig erzeugtem (Permutations-) Schlüssel

---

```
sage: # Remark: Enciphering here requires, that the length of msg is a multiple of keylen
sage: keylen = 14 # length of key
sage: A = AlphabeticStrings()
sage: T = TranspositionCryptosystem(A, keylen); T
Transposition cryptosystem on Free alphabetic string monoid on A-Z of block length 14
sage:
sage: P = "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 b"
sage: key = T.random_key(); key
(1,2,3,13,6,5,4,12,7)(11,14)
sage: msg = T.encoding(P); msg
ABCDEFGHIJKLMN O P Q R S T U V W X Y Z A B
sage: C = T.enciphering(key, msg); C
BCMLDEAHIJNGFKPQAZRSOVWXBUTY
sage: # decryption using the "deciphering method with key" instead of "enciphering with keyInv"
sage: DC = T.deciphering(key, C); DC
ABCDEFGHIJKLMN O P Q R S T U V W X Y Z A B
sage:
sage: # Just another way of decryption: Using "enciphering" with the inverse key
sage: keyInv = T.inverse_key(key); keyInv
(1,7,12,4,5,6,13,3,2)(11,14)
sage: DC = T.enciphering(keyInv, C); DC
ABCDEFGHIJKLMN O P Q R S T U V W X Y Z A B
sage:
sage: # Test correctness of decryption
sage: msg == DC
True
```

---

Das vierte Beispiel der Transpositions-Chiffren gibt zusätzlich die Größe des Schlüsselraums einer einfachen Spaltentransposition aus.

---

**Sage-Beispiel 2.4** Einfache Spalten-Transposition (mit Ausgabe der Größe des Schlüsselraumes)

---

```
sage: keylen = 14      # length of key
sage: A = AlphabeticStrings()
sage: T = TranspositionCryptosystem(A, keylen); T
Transposition cryptosystem on Free alphabetic string monoid on A-Z of block length 14
sage: T.key_space()
Symmetric group of order 14! as a permutation group
sage: # Remark: The key space is not quite correct as also permutations shorter than keylen are counted.
sage:
sage: P = "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 b"
sage: key = T.random_key(); key
(1,2,7)(3,9)(4,5,10,12,8,13,11)(6,14)
sage: msg = T.encoding(P); msg
ABCDEFGHIJKLMNPOQRSTUVWXYZAB
sage:
sage: # enciphering in one and in two steps
sage: C = T.enciphering(key, msg); C
BGIEJNAMCLDHKFPUWSXBOAQZRVYT
sage:
sage: enc = T(key); enc.key()
(1,2,7)(3,9)(4,5,10,12,8,13,11)(6,14)
sage: C = enc(msg); C
BGIEJNAMCLDHKFPUWSXBOAQZRVYT
sage:
sage: # deciphering
sage: DC = T.deciphering(key, C); DC
ABCDEFGHIJKLMNPOQRSTUVWXYZAB
```

---

## 2.5.2 Substitutions-Chiffren

Substitutions-Verschlüsselungen sind in Sage in der folgenden Klasse implementiert

```
sage.crypto.classical.SubstitutionCryptosystem
```

Das folgende Programmbeispiel lässt Sage eine Substitutions-Chiffre mit zufälligem Schlüssel erstellen. Den zufälligen Schlüssel kann man mit der Methode `random_key()` der Klasse `SubstitutionCryptosystem` erzeugen. Unterschiedliche Schlüssel erzeugen eine unterschiedliche Substitutions-Chiffre. Deshalb erhält man mit jedem Aufruf von `random_key()` voraussichtlich ein anderes Ergebnis.

---

### Sage-Beispiel 2.5 Monoalphabetische Substitution mit zufällig erzeugtem Schlüssel

---

```
sage: # plaintext/ciphertext alphabet
sage: A = AlphabeticStrings()
sage: S = SubstitutionCryptosystem(A)
sage:
sage: P = "Substitute this with something else better."
sage: key = S.random_key(); key
INZDHFUXJPATQOYLKSWGVECMRB
sage:
sage: # method encoding can be called from A or from T
sage: msg = A.encoding(P); msg
SUBSTITUTETHISWITHSOMETHINGELSEBETTER
sage: C = S.encrypting(key, msg); C
WVNWVGJGVGHGXJWCJGXWYQHGXJOUHTWHNHGGHS
sage:
sage: # We now decrypt the ciphertext to recover our plaintext.
sage:
sage: DC = S.decrypting(key, C); DC
SUBSTITUTETHISWITHSOMETHINGELSEBETTER
sage: msg == DC
True
```

---

### 2.5.3 Caesar-Chiffre

Das folgende Programmbeispiel erzeugt eine Caesar-Chiffre.

---

**Sage-Beispiel 2.6** Caesar (Substitution durch Shiften des Alphabets; Schlüssel explizit gegeben; Schritt-für-Schritt-Ansatz)

---

```
sage: # plaintext/ciphertext alphabet
sage: A = AlphabeticStrings()
sage: P = "Shift the alphabet three positions to the right."
sage:
sage: # construct Caesar cipher
sage: S = SubstitutionCryptosystem(A)
sage: key = A([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, \
....:         20, 21, 22, 23, 24, 25, 0, 1, 2])
sage:
sage: # encrypt message
sage: msg      = A.encoding(P); msg
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage: encrypt = S(key); encrypt
DEFGHIJKLMNOPQRSTUVWXYZABC
sage: C      = encrypt(msg); C
VKLIWWKHDOSKDEHWWKUHHSRVLWLRQVWRWKHULJKW
sage:
sage: # Next, we recover the plaintext.
sage: # decrypt message
sage: keyInv = A([23, 24, 25, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, \
....:         14, 15, 16, 17, 18, 19, 20, 21, 22])
sage: decrypt = S(keyInv); decrypt
XYZABCDEFGHIJKLMNQRSTUUVW
sage: DC      = decrypt(C); DC
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage: msg == DC
True
```

---

Das zweite Caesar-Beispiel macht dasselbe, aber der Programmcode ist variabler.

---

**Sage-Beispiel 2.7** Caesar (Substitution durch Shiften des Alphabets; Substitutions-Schlüssel wird berechnet)

---

```
sage: # plaintext/ciphertext alphabet
sage: A = AlphabeticStrings()
sage: keylen = len(A.gens()); keylen
26
sage: shift = 3
sage: P = "Shift the alphabet three positions to the right."
sage:
sage: # construct Caesar cipher
sage: S = SubstitutionCryptosystem(A)
sage: S
Substitution cryptosystem on Free alphabetic string monoid on A-Z
sage: # key = A([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, \
sage: #           20, 21, 22, 23, 24, 25, 0, 1, 2])
sage: key = [(i+shift).mod(keylen) for i in range(keylen)];
sage: key = A(key); key
DEFGHIJKLMNOPQRSTUVWXYZABC
sage: len(key)
26
sage:
sage: # encrypt message
sage: msg = A.encoding(P); msg
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage: C = S.enciphering(key, msg); C
VKLIWWKHDOSKDEHWWKUHHRSVLWLRQVWRWKHULJKW
sage:
sage: # Next, we recover the plaintext.
sage: # decrypt message
sage: # keyInv = A([23, 24, 25, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, \
sage: #           14, 15, 16, 17, 18, 19, 20, 21, 22])
sage: shiftInv=keylen-shift;
sage: keyInv = [(i+shiftInv).mod(keylen) for i in range(keylen)];
sage: keyInv = A(keyInv); keyInv
XYZABCDEFGHIJKLMNPOQRSTUVWXYZ
sage: DC = S.enciphering(keyInv, C); DC
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage:
sage: # Just another way of decryption: Using "deciphering" with the key
sage: DC = S.deciphering(key, C); DC
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage:
sage: msg == DC
True
```

---

### 2.5.4 Verschiebe-Chiffre

Die Verschiebe-Chiffre kann man sich auch als Verallgemeinerung der ursprünglichen Caesar-Chiffre denken: Caesar shiftete das Alphabet immer genau um 3 Positionen. Die Shift-Chiffre kann um eine beliebige Anzahl Positionen entlang des Alphabets shiften. Shiften ist eine spezielle Form der Substitution.

In den obigen Beispielen wurde das `SubstitutionCryptosystem` angewandt, und Caesar wurde als Sonderfall der Substitution implementiert. Andererseits kann man Caesar auch Spezialfall der Shift-Chiffre betrachten.

Die Shift-Chiffre ist direkt implementiert in der Sage-Klasse

```
sage.crypto.classical.ShiftCryptosystem
```

Im folgenden Beispiel konstruieren wir eine Shift-Chiffre über den Großbuchstaben des lateinischen Alphabets. Dann verschlüsseln wir den Klartext P durch Shiften um 12 Positionen. Schließlich entschlüsseln wir wieder das Chiffre C und überprüfen, dass das Ergebnis (DC) identisch mit dem ursprünglichen Klartext ist.

---

#### Sage-Beispiel 2.8 Verschiebe-Chiffre (über dem Großbuchstabenalphabet)

---

```
sage: # construct Shift cipher directly
sage: shiftcipher = ShiftCryptosystem(AlphabeticStrings()); shiftcipher
Shift cryptosystem on Free alphabetic string monoid on A-Z
sage: P = shiftcipher.encoding("Shift me any number of positions."); P
SHIFTMEANYNUMBEROFPOSITIONS
sage: key = 12 # shift can be any integer number
sage: # shift the plaintext by 12 positions to get the ciphertext
sage: C = shiftcipher.enciphering(key, P); C
ETURFYQMZKZGYNQDARBAEUFUAZE
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = shiftcipher.deciphering(key, C); DC
SHIFTMEANYNUMBEROFPOSITIONS
sage: DC == P
True
```

---

Die ursprüngliche Caesar-Chiffre ist eine Shift-Chiffre mit festem Schlüssel 3. Im nächsten Beispiel wird die Caesar-Chiffre über den Großbuchstaben des lateinischen Alphabets gebaut.

---

#### Sage-Beispiel 2.9 Caesar-Verschlüsselung mit der Verschiebe-Chiffre

---

```
sage: caesarcipher = ShiftCryptosystem(AlphabeticStrings())
sage: P = caesarcipher.encoding("Shift the alphabet by three positions to the right."); P
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage:
sage: key = 3 # shift the plaintext by exactly 3 positions
sage: C = caesarcipher.enciphering(key, P); C
VKLIWWKHDSKDEHWEBWKUHSRVLWLRQVWRWKHULJKW
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = caesarcipher.deciphering(key, C); DC
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage: DC == P
True
```

---



### 2.5.5 Affine Chiffren

Die affine Chiffre ist implementiert in the Sage-Klasse

`sage.crypto.classical.AffineCryptosystem`

Im folgenden Beispiel konstruieren wir eine affine Chiffre mit dem Schlüssel  $(3, 13)$  und benutzen sie, um einen gegebenen Klartext  $P$  zu verschlüsseln. Der Klartext wird dann wieder entschlüsselt und das Ergebnis  $DC$  mit dem ursprünglichen Klartext verglichen.

---

**Sage-Beispiel 2.10** Affine Chiffre mit dem Schlüssel  $(3, 13)$ 

---

```
sage: # create an affine cipher
sage: affineCipher = AffineCryptosystem(AlphabeticStrings()); affineCipher
Affine cryptosystem on Free alphabetic string monoid on A-Z
sage: P = affineCipher.encoding("The affine cryptosystem.")
sage: P
THEAFFINECRYPTOSYSTEM
sage:
sage: # encrypt the plaintext using the key (3, 13)
sage: a, b = (3, 13)
sage: C = affineCipher.enciphering(a, b, P)
sage: C
SIZNCCLAZTMHGSDPHPSZX
sage:
sage: # decrypt the ciphertext and make sure that it is equivalent to the original plaintext
sage: DC = affineCipher.deciphering(a, b, C)
sage: DC
THEAFFINECRYPTOSYSTEM
sage: DC == P
True
```

---

Man kann die Shift-Chiffre auch als Sonderfall der affinen Chiffre sehen. Dazu muss man den Schlüssel der affinen Chiffre auf die folgende Form  $(1, b)$  einschränken, wobei  $b$  jede ganze, nicht-negative Zahl sein kann. Das Sage-Beispiel 2.8 auf Seite 42 kann man wie folgt darstellen:

---

**Sage-Beispiel 2.11** Verschiebe-Chiffre (als Sonderfall der affinen Chiffre)

---

```
sage: # construct a shift cipher
sage: shiftcipher = AffineCryptosystem(AlphabeticStrings()); shiftcipher
Affine cryptosystem on Free alphabetic string monoid on A-Z
sage: P = shiftcipher.encoding("Shift me any number of positions.")
sage: P
SHIFTMEANYNUMBEROFPOSITIONS
sage:
sage: # shift the plaintext by 12 positions to get the ciphertext
sage: a, b = (1, 12)
sage: C = shiftcipher.enciphering(a, b, P)
sage: C
ETURFYQMZKZGYNQDARBAEUFUAZE
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = shiftcipher.deciphering(a, b, C); P
SHIFTMEANYNUMBEROFPOSITIONS
sage: DC == P
True
```

---

Man kann mit der affinen Chiffre auch eine Caesar-Chiffre bauen. Dazu muss der Ver-/Entschlüsselungsschlüssel den Wert  $(1, 3)$  haben. Das Sage-Beispiel 2.9 auf Seite 42 kann mit der affinen Chiffre folgendermaßen dargestellt werden.

---

**Sage-Beispiel 2.12** Caesar-Chiffre (als Sonderfall der affinen Chiffre)

---

```
sage: # create a Caesar cipher
sage: caesarcipher = AffineCryptosystem(AlphabeticStrings())
sage: P = caesarcipher.encoding("Shift the alphabet by three positions to the right.")
sage: P
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage:
sage: # shift the plaintext by 3 positions
sage: a, b = (1, 3)
sage: C = caesarcipher.enciphering(a, b, P)
sage: C
VKLIWWKHDOSKDEHWEBWKUHHSRVLWLRQVWRWKHULJKW
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = caesarcipher.deciphering(a, b, C)
sage: DC
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage: DC == P
True
```

---

## 2.5.6 Substitutions-Chiffre mit Symbolen

Im folgenden Sage-Programmbeispiel sind die Alphabetzeichen aus dem Binärsystem. Eine monoalphabetische Substitution über einem Binäralphabet bietet nur wenig Sicherheit: Weil das Klartext-/Geheintext-Alphabet nur aus den 2 Elementen 0 and 1 besteht, gibt es nur zwei mögliche Schlüssel: (0 1) and (1 0).

Anmerkung: Im Schlüssel einer allgemeinen Substitutions-Chiffre müssen alle Alphabetzeichen genau einmal auftreten.

---

### Sage-Beispiel 2.13 Monoalphabetische Substitution über dem Binär-Alphabet

---

```
sage: # the plaintext/ciphertext alphabet
sage: B = BinaryStrings()
sage: # substitution cipher over the alphabet B; no keylen argument possible
sage: S = SubstitutionCryptosystem(B); S
Substitution cryptosystem on Free binary string monoid
sage: # To get a substitute for each symbol, key has always the length of the alphabet
sage: key = S.random_key(); key
10
sage: len(key)
2
sage: P = "Working with binary numbers."
sage: # encryption
sage: msg = B.encoding(P); msg
01010111011011110111001001101011011010010110111001100111001000000111011101101\
00101110100011010000010000001100010011010010110111001100001011100100111100100\
1000000110111001110101011011010110001001100101011100100111001100101110
sage: C = S.enciphering(key, msg); C
1010100010010000100011011001010010010110100100011001100011011111000100010010\
110100010111001011110111110011101100101101001000110011110100011011000011011\
0111111001000110001010100100101001110110011010100011011000110011010001
sage: # decryption
sage: DC = S.deciphering(key, C); DC
01010111011011110111001001101011011010010110111001100111001000000111011101101\
00101110100011010000010000001100010011010010110111001100001011100100111100100\
10000001101110011101010110110110001001100101011100100111001100101110
sage: msg == DC
True
```

---

Anmerkung: Im Moment hat `S` kein Attribut `key`, und ich fand keine Möglichkeit, den Binärstring `DC` wieder in ASCII zurück zu verwandeln.

Das zweite Beispiel einer monoalphabetischen Substitution mit Symbolen benutzt ein größeres Alphabet für den Klartext-/Geheimtext-Zeichenraum als das erste Beispiel. Hier wird nun das hexadezimale Zahlensystem als Substitutions-Alphabet verwendet.

---

**Sage-Beispiel 2.14** Monoalphabetische Substitution über dem Hexadezimal-Alphabet (Dekodieren in Python)

---

```
sage: A = HexadecimalStrings()
sage: S = SubstitutionCryptosystem(A)
sage: key = S.random_key(); key
2b56a4e701c98df3
sage: len(key)
16
sage: # Number of possible keys
sage: factorial(len(key))
20922789888000
sage: P = "Working with a larger alphabet."
sage:
sage: msg = A.encoding(P); msg
576f726b696e6720776974682061206c617267657220616c7068616265742e
sage: C = S.enciphering(key, msg); C
47e375e9e1efe75277e17ae052eb52e8eb75e7e47552ebe872e0ebe5e47a5f
sage: DC = S.deciphering(key, C); DC
576f726b696e6720776974682061206c617267657220616c7068616265742e
sage: msg == DC
True
sage:
sage: # Conversion hex back to ASCII:
sage: # - AlphabeticStrings() and HexadecimalStrings() don't have according methods.
sage: # - So we used Python directly.
sage: import binascii
sage: DDC = binascii.a2b_hex(repr(DC)); DDC
'Working with a larger alphabet.'
sage:
sage: P == DDC
True
```

---

## 2.5.7 Vigenère-Verschlüsselung

Die Vigenère-Verschlüsselung ist in der folgenden Sage-Klasse implementiert

```
sage.crypto.classical.VigenereCryptosystem
```

Als Klartext-/Geheimtext-Zeichenraum kann man die lateinischen Großbuchstaben, das Binärsystem, das Oktalsystem oder Hexadezimalsystem wählen. Hier ist ein Beispiel, das mit der Klasse `AlphabeticStrings` die Großbuchstaben nutzt.

---

### Sage-Beispiel 2.15 Vigenère-Verschlüsselung

---

```
sage: # construct Vigenere cipher
sage: keylen = 14
sage: A = AlphabeticStrings()
sage: V = VigenereCryptosystem(A, keylen); V
Vigenere cryptosystem on Free alphabetic string monoid on A-Z of period 14
sage:
sage: # alternative could be a given key: key = A('ABCDEFGHIJKLMN'); key
sage: key = V.random_key(); key
WSSSEEGVVAARUD
sage: len(key)
14
sage:
sage: # encoding
sage: P = "The Vigenere cipher is polyalphabetic."
sage: len(P)
38
sage: msg = V.encoding(P); msg      # alternative: msg = A.encoding(P); msg
THEVIGENERECIPHERISPOLYALPHABETIC
sage:
sage: # encryption [2 alternative ways (in two steps or in one): both work]
sage: # encrypt = V(key); encrypt
sage: # C = encrypt(msg); C
sage: C = V.enciphering(key, msg); C
PZWNMKKIZRETCSJDWJAWTUGTALGBDXWLAG
sage:
sage: # decryption
sage: DC = V.deciphering(key, C); DC
THEVIGENERECIPHERISPOLYALPHABETIC
sage: msg == DC
True
```

---

## 2.5.8 Hill-Verschlüsselung

Die Hill [Hill1929, Hill1931]- oder Matrix-Verschlüsselung ist mathematisch anspruchsvoller als die Verfahren in den bisherigen Programmbeispielen dieses Kapitels. Der Schlüssel dieses Verfahrens ist eine invertierbare quadratische Matrix. Auch der Klar-/Geheimtext wird als Matrix verarbeitet. Die Ver- und Entschlüsselungsprozesse nutzen Matrizen-Operationen modulo 26. Die Hill-Verschlüsselung ist implementiert in der Sage-Klasse

```
sage.crypto.classical.HillCryptosystem
```

Im folgenden Beispiel besteht der Klar-/Geheimtext-Zeichenraum wieder aus den Großbuchstaben des lateinischen Alphabets. Im Hill-Verfahren wird jedem Alphabet-Zeichen eine eindeutige natürliche Zahl modulo 26 zugewiesen. Die Größe der Matrix (auch Matrix-Dimension genannt) ist vom Hill-Verfahren selbst nicht beschränkt.

**Bemerkung:** Vergleich der Hill-Implementierung in CrypTool v1.4.30 und in Sage v4.2.1:

- Sage bietet schnelle Operationen auf der Kommandozeile; CrypTool stellt seine Funktionalität nur in einer GUI zur Verfügung.
- Sage bietet jede Matrix-Dimension an; CrypTool ist beschränkt auf die Werte 1 bis 10.
- Sage weist dem ersten Alphabet-Zeichen immer den Wert 0 zu; und Sage benutzt die Multiplikationsvariante Klartext-Zeilenvektor \* Schlüsselmatrix:  $C = P * M$ .
- CrypTool erlaubt die Wahl zwischen 0 und 1 für das erste Alphabet-Zeichen; und es bietet auch die umgekehrte Multiplikationsvariante.

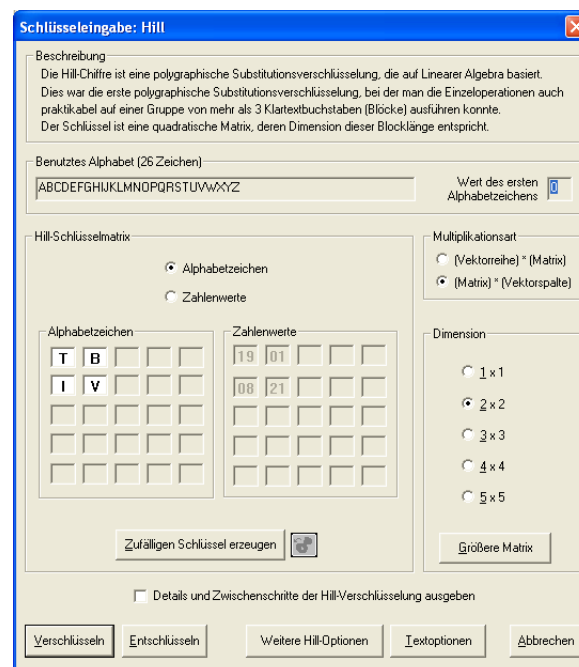


Abbildung 2.2: Hill-Dialog in CrypTool mit den verfügbaren Operationen und Optionen

---

## Sage-Beispiel 2.16 Hill-Verschlüsselung

---

```
sage: # construct a Hill cipher
sage: keylen = 19      # keylen = 3  # Alternative key length with non-random small key
sage: A = AlphabeticStrings()
sage: H = HillCryptosystem(A, keylen); H
Hill cryptosystem on Free alphabetic string monoid on A-Z of block length 19
sage:
sage: # To create key non-randomly, HKS is necessary [even H.key_space() is not enough].
sage: # HKS = H.key_space()
sage: # key = HKS([[1,0,1],[0,1,1],[2,2,3]]); key
sage:
sage: # Random key creation
sage: key = H.random_key(); key
[10 7 5 2 0 6 10 23 15 7 17 19 18 2 9 12 0 10 11]
[23 1 1 10 4 9 21 1 25 22 19 8 17 22 15 8 12 25 22]
[ 4 12 16 15 1 12 24 5 9 13 5 15 8 21 23 24 22 20 6]
[ 5 11 6 7 3 12 8 9 21 20 9 4 16 18 10 3 2 23 18]
[ 8 22 14 14 20 13 21 19 3 13 2 11 13 23 9 25 25 6 8]
[24 25 8 24 7 18 3 20 6 11 25 5 6 19 7 24 2 4 10]
[15 25 11 1 4 7 11 24 20 2 18 4 9 8 12 19 24 0 12]
[14 6 2 9 11 20 13 4 10 11 4 23 14 22 14 16 9 12 18]
[12 10 21 5 21 15 16 17 19 20 1 1 15 5 0 2 23 4 14]
[21 15 15 16 15 20 4 10 25 7 15 4 7 12 24 9 19 10 6]
[25 15 2 3 17 23 21 16 8 18 23 4 22 11 15 19 6 0 15]
[14 23 9 3 18 15 10 18 7 5 12 23 11 9 22 21 20 4 14]
[ 3 6 8 13 20 16 11 1 13 10 4 21 25 15 12 3 0 11 18]
[21 25 14 6 11 3 21 0 19 17 5 8 5 4 9 2 23 19 15]
[ 8 11 9 11 20 15 6 1 3 18 18 22 16 17 6 3 15 11 2]
[21 15 5 22 2 9 0 4 22 10 2 10 19 19 17 19 1 21 4]
[ 7 17 9 2 15 5 14 3 6 9 12 12 22 15 8 4 21 14 19]
[19 14 24 19 7 5 22 22 13 14 7 18 17 19 25 2 1 23 6]
[ 2 6 14 22 17 7 23 6 22 7 13 20 0 14 23 17 6 1 12]
sage:
sage: # encoding and encryption
sage: P = "Hill or matrix cipher uses matrix operations."
sage: len(P)
45
sage: # implementation requires: Length of msg is a multiple of matrix dimension (block_length)
sage: msg = H.encoding(P); msg
HILLORMATRIXCIPHERUSESMATRIXOPERATIONS
sage: len(msg)
38
sage:
sage: # encryption
sage: C = H.enciphering(key, msg); C
CRWCKPRVYXNBRZTNZCTQWFSDBCHABGMNEHVP
sage:
sage: # decryption
sage: DC = H.deciphering(key, C); DC
HILLORMATRIXCIPHERUSESMATRIXOPERATIONS
sage: msg == DC
True
sage:
sage: # alternative decryption using inverse matrix
sage: keyInv = H.inverse_key(key); keyInv
[ 6 23 1 23 3 12 17 22 6 16 22 14 18 3 1 10 21 16 20]
[18 23 15 25 24 23 7 4 10 7 21 7 9 0 13 22 5 5 23]
...
[10 11 12 6 11 17 13 9 19 16 14 24 4 8 5 16 18 20 1]
[19 16 16 21 1 19 7 12 3 18 1 17 7 10 24 21 7 16 11]
sage: DC = H.enciphering(keyInv, C); DC
HILLORMATRIXCIPHERUSESMATRIXOPERATIONS
```

# Literaturverzeichnis

- [ACA2002] American Cryptogram Association,  
*Length and Standards for all ACA Ciphers*,  
2002.  
<http://www.cryptogram.org/cdb/aca.info/aca.and.you/chap08.html#>
- [Bauer1995] Friedrich L. Bauer,  
*Entzifferte Geheimnisse*,  
Springer, 1995.
- [Bauer2000] Friedrich L. Bauer,  
*Decrypted Secrets*,  
Springer 1997, 2. Auflage 2000.
- [Crowley2000] Paul Crowley,  
*Mirdek: A card cipher inspired by „Solitaire“*,  
2000.  
<http://www.ciphergoth.org/crypto/mirdek/>
- [DA1999] Data encryption page des ThinkQuest Team 27158 für ThinkQuest 1999  
(Kein Update seit 1999, keine Suchmöglichkeit),  
1999.  
<http://library.thinkquest.org/27158/>
- [Goebel2003] Greg Goebel,  
*Codes, Ciphers and Codebreaking*,  
2003.  
<http://www.vectorsite.net/ttcode.htm>
- [Hill1929] Lester S. Hill,  
*Cryptography in an Algebraic Alphabet*,  
The American Mathematical Monthly, 36(6): 306–312, 1929.
- [Hill1931] Lester S. Hill,  
*Concerning Certain Linear Transformation Apparatus of Cryptography*,  
The American Mathematical Monthly, 38(3): 135–154, 1931.
- [Nichols1996] Randall K. Nichols,  
*Classical Cryptography Course, Volume 1 and 2*,  
Aegean Park Press 1996;  
oder in 12 Lektionen online unter  
<http://www.fortunecity.com/skyscraper/coding/379/lesson1.htm>



- [Nguyen2009] Minh Van Nguyen,  
*Exploring Cryptography Using the Sage Computer Algebra System*,  
 Victoria University, 2009,  
 Siehe Sage publications <http://www.sagemath.org/library-publications.html>,  
<http://www.sagemath.org/files/thesis/nguyen-thesis-2009.pdf>,  
<http://sites.google.com/site/nguyenminh2/honours-thesis-2009.pdf>
- [Savard1999] John J. G. Savard,  
*A Cryptographic Compendium*,  
 1999.  
<http://www.quadibloc.com/crypto/jsencrypt.htm>
- [Schmeh2004] Klaus Schmeh,  
*Die Welt der geheimen Zeichen. Die faszinierende Geschichte der Verschlüsselung*,  
 W3L Verlag Bochum, 1. Auflage 2004.
- [Schmeh2007] Klaus Schmeh,  
*Codeknacker gegen Codemacher. Die faszinierende Geschichte der Verschlüsselung*,  
 W3L Verlag Bochum, 2. Auflage 2007.  
 Dieses Buch ist das aktuellste unter denen, die sich mit der Geschichte der Kryptographie beschäftigen. Es enthält eine kleine Sammlung gelöster und ungelöster Kryptorätsel. Eine der Challenges nutzt den Doppelwürfel (double column transposition) mit zwei langen Schlüsseln, die auch unterschiedlich sind.
- [Schneier1999] Bruce Schneier,  
*The Solitaire Encryption Algorithm*,  
 Version 1.2, 1999.  
<http://www.schneier.com/solitaire.html>
- [Singh2001] Simon Singh,  
*Geheime Botschaften. Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet*,  
 dtv, 2001.
- [ThinkQuest1999] ThinkQuest Team 27158,  
*Data Encryption*,  
 1999.  
<http://library.thinkquest.org/27158/>

# Kapitel 3

## Primzahlen

(Bernhard Esslinger, Mai 1999; Updates: Nov. 2000, Dez. 2001, Juni 2003, Mai 2005, März 2006, Juni 2007, Januar 2010)

*Albert Einstein*<sup>1</sup>:  
Der Fortschritt lebt vom Austausch des Wissens.

### 3.1 Was sind Primzahlen?

Primzahlen sind ganze, positive Zahlen größer gleich 2, die man nur durch 1 und durch sich selbst teilen kann. Alle anderen natürlichen Zahlen größer gleich 2 lassen sich durch Multiplikation von Primzahlen bilden.

Somit bestehen die *natürlichen* Zahlen  $\mathbb{N} = \{1, 2, 3, 4, \dots\}$  aus

- der Zahl 1 (dem Einheitswert)
- den Primzahlen (primes) und
- den zusammengesetzten Zahlen (composite numbers).

Primzahlen haben aus 3 Gründen besondere Bedeutung erlangt:

- Sie werden in der Zahlentheorie als die Grundbausteine der natürlichen Zahlen betrachtet, anhand derer eine Menge genialer mathematischer Überlegungen geführt wurden.
- Sie haben in der modernen Kryptographie (Public-Key-Kryptographie) große praktische Bedeutung erlangt. Das verbreitetste Public-Key-Verfahren ist die Ende der siebziger Jahre erfundene RSA-Verschlüsselung. Nur die Verwendung (großer) Primzahlen für bestimmte Parameter garantiert die Sicherheit des Algorithmus sowohl beim RSA-Verfahren als auch bei noch moderneren Verfahren (digitale Signatur, Elliptische Kurven).
- Die Suche nach den größten bekannten Primzahlen hat wohl bisher keine praktische Verwendung, erfordert aber die besten Rechner, gilt als hervorragender Benchmark (Möglichkeit zur Leistungsbestimmung von Computern) und führt zu neuen Formen der Berechnungen auf mehreren Computern  
(siehe auch: <http://www.mersenne.org/prime.htm>).

---

<sup>1</sup>Albert Einstein, deutscher Physiker und Nobelpreisträger, 14.03.1879–14.04.1955.

Von Primzahlen ließen sich im Laufe der letzten zwei Jahrtausende sehr viele Menschen faszinieren. Der Ehrgeiz, zu neuen Erkenntnissen über Primzahlen zu gelangen, führte dabei oft zu genialen Ideen und Schlussfolgerungen. Im folgenden wird in einer leicht verständlichen Art in die mathematischen Grundlagen der Primzahlen eingeführt. Dabei klären wir auch, was über die Verteilung (Dichte, Anzahl von Primzahl in einem bestimmten Intervall) der Primzahlen bekannt ist oder wie Primzahltests funktionieren.

## 3.2 Primzahlen in der Mathematik

Jede ganze Zahl hat Teiler. Die Zahl 1 hat nur einen, nämlich sich selbst. Die Zahl 12 hat die sechs Teiler 1, 2, 3, 4, 6, 12. Viele Zahlen sind nur durch sich selbst und durch 1 teilbar. Bezüglich der Multiplikation sind dies die „Atome“ im Bereich der Zahlen. Diese Zahlen nennt man Primzahlen.

In der Mathematik ist eine etwas andere (aber äquivalente) Definition üblich.

**Definition 3.2.1.** Eine ganze Zahl  $p \in \mathbf{N}$  heißt Primzahl, wenn  $p > 1$  und  $p$  nur die trivialen Teiler  $\pm 1$  und  $\pm p$  besitzt.

Per definitionem ist die Zahl 1 keine Primzahl. Im weiteren bezeichnet der Buchstabe  $p$  stets eine Primzahl.

Die Primzahlenfolge startet mit

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,  $\dots$ .

Unter den ersten 100 Zahlen gibt es genau 25 Primzahlen. Danach nimmt ihr prozentualer Anteil stets ab. Primzahlen können nur auf eine einzige *triviale* Weise zerlegt werden:

$$5 = 1 \cdot 5, \quad 17 = 1 \cdot 17, \quad 1013 = 1 \cdot 1.013, \quad 1.296.409 = 1 \cdot 1.296.409.$$

Alle Zahlen, die 2 und mehr von 1 verschiedene Faktoren haben, nennt man *zusammengesetzte* Zahlen. Dazu gehören

$$4 = 2 \cdot 2, \quad 6 = 2 \cdot 3$$

aber auch Zahlen, die *wie Primzahlen aussehen*, aber doch keine sind:

$$91 = 7 \cdot 13, \quad 161 = 7 \cdot 23, \quad 767 = 13 \cdot 59.$$

**Satz 3.2.1.** Jede ganze Zahl  $m$  größer als 1 besitzt einen kleinsten Teiler größer als 1. Dieser ist eine Primzahl  $p$ . Sofern  $m$  nicht selbst eine Primzahl ist, gilt:  $p$  ist kleiner oder gleich der Quadratwurzel aus  $m$ .

Aus den Primzahlen lassen sich alle ganzen Zahlen größer als 1 zusammensetzen — und das sogar in einer eindeutigen Weise. Dies besagt der **1. Hauptsatz der Zahlentheorie** (= Hauptsatz der elementaren Zahlentheorie = fundamental theorem of arithmetic = fundamental building block of all positive integers).

**Satz 3.2.2.** Jedes Element  $n$  größer als 1 der natürlichen Zahlen lässt sich als Produkt  $n = p_1 \cdot p_2 \cdot \dots \cdot p_m$  von Primzahlen schreiben. Sind zwei solche Zerlegungen

$$n = p_1 \cdot p_2 \cdot \dots \cdot p_m = p'_1 \cdot p'_2 \cdot \dots \cdot p'_{m'}$$

gegeben, dann gilt nach eventuellem Umsortieren  $m = m'$  und für alle  $i$ :  $p_i = p'_i$ . ( $p_1, p_2, \dots, p_m$  nennt man die Primfaktoren von  $n$ ).

In anderen Worten: Jede natürliche Zahl außer der 1 lässt sich auf genau eine Weise als Produkt von Primzahlen schreiben, wenn man von der Reihenfolge der Faktoren absieht. Die Faktoren sind also eindeutig (die *Expansion in Faktoren* ist eindeutig)! Zum Beispiel ist

$$60 = 2 \cdot 2 \cdot 3 \cdot 5 = 2^2 \cdot 3^1 \cdot 5^1$$

Und das ist — bis auf eine veränderte Reihenfolge der Faktoren — die einzige Möglichkeit, die Zahl 60 in Primfaktoren zu zerlegen. Wenn man nicht nur Primzahlen als Faktoren zulässt, gibt es mehrere Möglichkeiten der Zerlegung in Faktoren und die **Eindeutigkeit** (uniqueness) geht verloren:

$$60 = 1 \cdot 60 = 2 \cdot 30 = 4 \cdot 15 = 5 \cdot 12 = 6 \cdot 10 = 2 \cdot 3 \cdot 10 = 2 \cdot 5 \cdot 6 = 3 \cdot 4 \cdot 5 = \dots$$

Der folgende Absatz wendet sich eher an die mit der mathematischen Logik vertrauteren Menschen: Der 1. Hauptsatz ist nur scheinbar selbstverständlich. Man kann viele andere Zahlenmengen (ungleich der positiven ganzen Zahlen größer als 1) konstruieren, bei denen selbst eine Zerlegung in die Primfaktoren dieser Mengen nicht eindeutig ist: In der Menge  $M = \{1, 5, 10, 15, 20, \dots\}$  gibt es unter der Multiplikation kein Analogon zum Hauptsatz. Die ersten fünf Primzahlen dieser Folge sind 5, 10, 15, 20, 30 (beachte: 10 ist prim, da innerhalb dieser Menge 5 kein Teiler von 10 ist — das Ergebnis 2 ist kein Element der gegebenen Grundmenge  $M$ ). Da in  $M$  gilt:

$$100 = 5 \cdot 20 = 10 \cdot 10$$

und sowohl 5, 10, 20 Primzahlen dieser Menge sind, ist hier die Zerlegung in Primfaktoren nicht eindeutig.

### 3.3 Wie viele Primzahlen gibt es?

Für die natürlichen Zahlen sind die Primzahlen vergleichbar mit den Elementen in der Chemie oder den Elementarteilchen in der Physik (vgl. [Blum1999, S. 22]).

Während es nur 92 natürliche chemische Elemente gibt, ist die Anzahl der Primzahlen unbegrenzt. Das wusste schon der Grieche Euklid<sup>2</sup> im dritten vorchristlichen Jahrhundert.

**Satz 3.3.1** (Euklid<sup>3</sup>). *Die Folge der Primzahlen bricht nicht ab, es gibt also unendlich viele Primzahlen.*

Sein Beweis, dass es unendlich viele Primzahlen gibt, gilt bis heute als ein Glanzstück mathematischer Überlegung und Schlussfolgerung (**Widerspruchsbeweis**). Er nahm an, es gebe nur endlich viele Primzahlen und damit eine größte Primzahl. Daraus zog er solange logische Schlüsse, bis er auf einen offensichtlichen Widerspruch stieß. Damit musste etwas falsch sein. Da sich in die Schlusskette kein Lapsus eingeschlichen hatte, konnte es nur die Annahme sein. Demnach musste es unendlich viele Primzahlen geben!

<sup>2</sup>Euklid, griechischer Mathematiker des 4./3. Jahrhunderts vor Christus. Wirkte an der Akademie in Alexandria und verfasste mit den „Elementen“ das bekannteste systematische Lehrbuch der griechischen Mathematik.

<sup>3</sup>Die üblich gewordene Benennung bedeutet nicht unbedingt, dass Euklid der Entdecker des Satzes ist, da dieser nicht bekannt ist. Der Satz wird bereits in Euklids „Elementen“ (Buch IX, Satz 20) formuliert und bewiesen. Die dortige Formulierung ist insofern bemerkenswert, als sie das Wort „unendlich“ nicht verwendet; sie lautet

*Οἱ πρῶτοι ἀριθμοὶ πλείους εἰσὶ παντὸς τοῦ προτεθέντος πλήθους πρῶτων ἀριθμῶν,*

zu deutsch: Die Primzahlen sind mehr als jede vorgegebene Menge von Primzahlen.

**Euklid's Widerspruchsbeweis** führt die Argumentation wie folgt:

**Beweis**

**Annahme:** Es gibt *endlich* viele Primzahlen.

**Schluss:** Dann lassen sie sich auflisten  $p_1 < p_2 < p_3 < \dots < p_n$ , wobei  $n$  für die (endliche) Anzahl der Primzahlen steht.  $p_n$  wäre also die größte Primzahl. Nun betrachtet Euklid die Zahl  $a = p_1 \cdot p_2 \cdot \dots \cdot p_n + 1$ . Diese Zahl kann keine Primzahl sein, da sie in unserer Primzahlenliste nicht auftaucht. Also muss sie durch eine Primzahl teilbar sein. D.h. es gibt eine natürliche Zahl  $i$  zwischen 1 und  $n$ , so dass  $p_i$  die Zahl  $a$  teilt. Natürlich teilt  $p_i$  auch das Produkt  $a - 1 = p_1 \cdot p_2 \cdot \dots \cdot p_n$ , da  $p_i$  ja ein Faktor von  $a - 1$  ist. Da  $p_i$  die Zahlen  $a$  und  $a - 1$  teilt, teilt sie auch die Differenz dieser Zahlen. Daraus folgt:  $p_i$  teilt  $a - (a - 1) = 1$ .  $p_i$  müsste also 1 teilen und das ist unmöglich.

**Widerspruch:** Unsere Annahme war falsch.

Also gibt es *unendlich* viele Primzahlen (siehe Übersicht unter 3.9 über die Anzahl von Primzahlen in verschiedenen Intervallen).  $\square$

Wir erwähnen hier auch noch eine andere, auf den ersten Blick überraschende Tatsache, dass nämlich in der Folge aller Primzahlen  $p_1, p_2, \dots$  Lücken von beliebig großer Länge  $n$  auftreten. Unter den  $n$  aufeinanderfolgenden natürlichen Zahlen

$$(n+1)! + 2, \dots, (n+1)! + (n+1),$$

ist keine eine Primzahl, da ja in ihnen der Reihe nach die Zahlen  $2, \dots, n+1$  als echte Teiler enthalten sind (Dabei bedeutet  $n!$  das Produkt der ersten  $n$  natürlichen Zahlen, also  $n! = n * (n-1) * \dots * 3 * 2 * 1$ ).

## 3.4 Die Suche nach sehr großen Primzahlen

Die größten heute bekannten Primzahlen haben mehrere Millionen Stellen. Das ist unvorstellbar groß. Die Anzahl der Elementarteilchen im Universum wird auf eine „nur“ 80-stellige Zahl geschätzt (siehe Übersicht unter 3.11 über verschiedene Größenordnungen / Dimensionen).

### 3.4.1 Die 20+ größten bekannten Primzahlen (Stand Juli 2009)

In der folgenden Tabelle sind die größten bekanntesten Primzahlen und eine Beschreibung des jeweiligen Zahlentyps aufgeführt<sup>4</sup>:

---

<sup>4</sup>Eine jeweils aktuelle Fassung findet sich im Internet unter <http://primes.utm.edu/largest.html> und <http://primes.utm.edu/merzenne/index.html>.

	Definition	Dezimalstellen	Wann	Beschreibung
1	$2^{43.112.609} - 1$	12.978.189	2008	Mersenne, 47. bekannte
2	$2^{42.643.801} - 1$	12.837.064	2009	Mersenne, 46. bekannte
3	$2^{37.156.667} - 1$	11.185.272	2008	Mersenne, 45. bekannte
4	$2^{32.582.657} - 1$	9.808.358	2006	Mersenne, 44. bekannte
5	$2^{30.402.457} - 1$	9.152.052	2005	Mersenne, 43. bekannte
6	$2^{25.964.951} - 1$	7.816.230	2005	Mersenne, 42. bekannte
7	$2^{24.036.583} - 1$	7.235.733	2004	Mersenne, 41. bekannte
8	$2^{20.996.011} - 1$	6.320.430	2003	Mersenne, 40. bekannte
9	$2^{13.466.917} - 1$	4.053.946	2001	Mersenne, M-39
10	$19.249 \cdot 2^{13.018.586} + 1$	3.918.990	2007	Verallgem. Mersenne <sup>5</sup>
11	$27.653 \cdot 2^{9.167.433} + 1$	2.759.677	2005	Verallgem. Mersenne
12	$28.433 \cdot 2^{7.830.457} + 1$	2.357.207	2004	Verallgem. Mersenne
13	$2^{6.972.593} - 1$	2.098.960	1999	Mersenne, M-38
14	$5.359 \cdot 2^{5.054.502} + 1$	1.521.561	2003	Verallgem. Mersenne
15	$4.847 \cdot 2^{3.321.063} + 1$	999.744	2005	Verallgem. Mersenne
16	$3 \cdot 2^{3.136.255} - 1$	944.108	2007	Verallgem. Mersenne
17	$2^{3.021.377} - 1$	909.526	1998	Mersenne, M-37
18	$2^{2.976.221} - 1$	895.932	1997	Mersenne, M-36
19	$222.361 \cdot 2^{2.854.840} + 1$	859.398	2006	Verallgem. Mersenne
20	$1.372.930^{131.072} + 1$	804.474	2003	Verallgem. Fermat <sup>6</sup>
21	$1.361.244^{131.072} + 1$	803.988	2004	Verallgem. Fermat
22	$1.176.694^{131.072} + 1$	795.695	2003	Verallgem. Fermat
23	$342.673 \cdot 2^{2.639.439} - 1$	794.556	2007	Verallgem. Mersenne

Tabelle 3.1: Die 20+ größten Primzahlen und ihr jeweiliger Zahlentyp (Stand Juli 2009)

Die größte bekannte Primzahl ist eine Mersenne-Primzahl. Diese wurde vom GIMPS-Projekt (Kapitel 3.4.2) gefunden.

Unter den größten bekannten Primzahlen befinden sich außerdem Zahlen vom Typ verallgemeinerte Mersennezahl (Kapitel 3.6.2) und vom Typ verallgemeinerte Fermatzahl (Kapitel 3.6.5).

<sup>6</sup>Diese Zahl wurde am 26.3.2007 im verteilt rechnenden Internet-Projekt „Seventeen or Bust“ (SoB) (<http://www.seventeenorbust.com>) gefunden. Anders als das bekannte GIMPS-Projekt (Kapitel 3.4.2), das immer größere der unendlich vielen Primzahlen aufspürt, könnte SoB aber irgendwann mal die gesetzte Aufgabe vollständig erledigt haben.

Das SoB-Projekt versucht rechnerisch zu beweisen, dass die Zahl  $k = 78.557$  die kleinste Sierpinski-Zahl ist (John Selfridge bewies 1962, dass 78.557 eine Sierpinski-Zahl ist).

Der berühmte polnische Mathematiker Waclaw Sierpinski (1882 bis 1969) hatte im Jahre 1960 nachgewiesen, dass es unendlich viele ungerade natürliche Zahlen  $k$  gibt, die folgende Eigenschaft erfüllen: Für jede Sierpinski-Zahl  $k$  gilt: Sämtliche Zahlen  $N = k \cdot 2^n + 1$  sind für alle natürlichen  $n \geq 1$  zusammengesetzte Zahlen (Sierpinski's Composite Number Theorem, <http://mathworld.wolfram.com/SierpinskisCompositeNumberTheorem.html>).

Am Projektanfang im Jahre 2002 gab es noch 17 mögliche Kandidaten  $< 78557$  (daher der Name des Projekts „Seventeen or Bust“). Es reicht, wenn man ein einziges Gegenbeispiel findet, um einen Kandidaten  $k$  auszuschließen, also ein einziges  $n \geq 1$  zu finden, so dass  $N = k \cdot 2^n + 1$  prim ist. Dass bei dieser Suche sehr große Primzahlen gefunden werden, ist also eigentlich nur ein „Nebenprodukt“ der Aufgabenstellung.

Vergleiche auch: <http://www.heise.de/newsticker/meldung/89582>.

<sup>6</sup>Verallgemeinerte Fermatzahl:  $1.372.930^{131.072} + 1 = 1.372.930^{(2^{17})} + 1$

### 3.4.2 Spezielle Zahlentypen – Mersennezahlen und Mersenne-Primzahlen

Nahezu alle bekannten riesig großen Primzahlen sind spezielle Kandidaten, sogenannte *Mersennezahlen*<sup>7</sup> der Form  $2^p - 1$ , wobei  $p$  eine Primzahl ist. Nicht alle Mersennezahlen sind prim:

$$\begin{aligned}2^2 - 1 &= 3 && \Rightarrow \text{prim} \\2^3 - 1 &= 7 && \Rightarrow \text{prim} \\2^5 - 1 &= 31 && \Rightarrow \text{prim} \\2^7 - 1 &= 127 && \Rightarrow \text{prim} \\2^{11} - 1 &= 2.047 = 23 \cdot 89 && \Rightarrow \text{NICHT prim!}\end{aligned}$$

Dass Mersennezahlen nicht immer Primzahlen (Mersenne-Primzahlen) sind, wusste auch schon Mersenne (siehe Exponent  $p = 11$ ). Eine Mersennezahl, die prim ist, wird Mersenne-Primzahl genannt.

Dennoch ist ihm der interessante Zusammenhang zu verdanken, dass eine Zahl der Form  $2^n - 1$  keine Primzahl sein kann, wenn  $n$  eine zusammengesetzte Zahl ist:

**Satz 3.4.1** (Mersenne). *Wenn  $2^n - 1$  eine Primzahl ist, dann folgt,  $n$  ist ebenfalls eine Primzahl (oder anders formuliert:  $2^n - 1$  ist nur dann prim, wenn  $n$  prim ist).*

#### Beweis

Der Beweis des Satzes von Mersenne kann durch Widerspruch durchgeführt werden. Wir nehmen also an, dass es eine zusammengesetzte natürliche Zahl  $n$  mit echter Zerlegung  $n = n_1 \cdot n_2$  gibt, mit der Eigenschaft, dass  $2^n - 1$  eine Primzahl ist.

Wegen

$$\begin{aligned}(x^r - 1)((x^r)^{s-1} + (x^r)^{s-2} + \dots + x^r + 1) &= ((x^r)^s + (x^r)^{s-1} + (x^r)^{s-2} + \dots + x^r) \\&\quad - ((x^r)^{s-1} + (x^r)^{s-2} + \dots + x^r + 1) \\&= (x^r)^s - 1 = x^{rs} - 1,\end{aligned}$$

folgt

$$2^{n_1 n_2} - 1 = (2^{n_1} - 1)((2^{n_1})^{n_2-1} + (2^{n_1})^{n_2-2} + \dots + 2^{n_1} + 1).$$

Da  $2^n - 1$  eine Primzahl ist, muss einer der obigen beiden Faktoren auf der rechten Seite gleich 1 sein. Dies kann nur dann der Fall sein, wenn  $n_1 = 1$  oder  $n_2 = 1$  ist. Dies ist aber ein Widerspruch zu unserer Annahme. Deshalb ist unsere Annahme falsch. Also gibt es keine zusammengesetzte Zahl  $n$ , so dass  $2^n - 1$  eine Primzahl ist.  $\square$

Leider gilt dieser Satz nur in einer Richtung (die Umkehrung gilt nicht, keine Äquivalenz): das heißt, dass es prime Exponenten gibt, für die die zugehörige Mersennezahl **nicht** prim ist (siehe das obige Beispiel  $2^{11} - 1$ , wo 11 prim ist, aber  $2^{11} - 1$  nicht).

Mersenne behauptete, dass  $2^{67} - 1$  eine Primzahl ist. Auch zu dieser Behauptung gibt es eine interessante mathematische Historie: Zuerst dauerte es über 200 Jahre, bis Edouard Lucas (1842-1891) bewies, dass diese Zahl zusammengesetzt ist. Er argumentierte aber indirekt und kannte keinen der Faktoren. Dann zeigte Frank Nelson Cole<sup>8</sup> 1903, aus welchen Faktoren diese Primzahl besteht:

$$2^{67} - 1 = 147.573.952.589.676.412.927 = 193.707.721 \cdot 761.838.257.287.$$

<sup>7</sup>Marin Mersenne, französischer Priester und Mathematiker, 08.09.1588–01.09.1648.

<sup>8</sup>Frank Nelson Cole, amerikanischer Mathematiker, 20.09.1861–26.05.1926.

Er gestand, 20 Jahre an der Faktorisierung (Zerlegung in Faktoren)<sup>9</sup> dieser 21-stelligen Dezimalzahl gearbeitet zu haben!

Dadurch, dass man bei den Exponenten der Mersennezahlen nicht alle natürlichen Zahlen verwendet, sondern nur die Primzahlen, engt man den *Versuchsraum* deutlich ein. Die derzeit bekannten Mersenne-Primzahlen gibt es für die Exponenten<sup>10</sup>

2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1.279, 2.203, 2.281, 3.217, 4.253,  
4.423, 9.689, 9.941, 11.213, 19.937, 21.701, 23.207, 44.497, 86.243, 110.503, 132.049,  
216.091, 756.839, 859.433, 1.257.787, 1.398.269, 2.976.221, 3.021.377, 6.972.593,  
13.466.917, 20.996.011, 24.036.583, 25.964.951, 30.402.457, 32.582.657,  
37.156.667, 43.112.609, 42.643.801.

Damit sind heute 47 Mersenne-Primzahlen bekannt.

Die 19. Zahl mit dem Exponenten 4.253 war die erste mit mindestens 1.000 Stellen im Zehnersystem (der Mathematiker Samuel Yates prägte dafür den Ausdruck *titanische* Primzahl; sie wurde 1961 von Hurwitz gefunden); die 27. Zahl mit dem Exponenten 44.497 war die erste mit mindestens 10.000 Stellen im Zehnersystem (Yates prägte dafür den Ausdruck *gigantische* Primzahl. Diese Bezeichnungen sind heute längst veraltet).

Für die ersten 39 Mersenne-Primzahlen weiß man inzwischen, dass diese Liste vollständig ist. Die Exponenten bis zur 40. bekannten Mersenne-Primzahl sind noch nicht vollständig geprüft<sup>11</sup>.

### M-37 – Januar 1998

Die 37. Zahl Mersenne-Primzahl,

$$2^{3.021.377} - 1$$

wurde im Januar 1998 gefunden und hat 909.526 Stellen im Zehnersystem, was 33 Seiten in der FAZ entspricht!

### M-38 – Juni 1999

Die 38. Mersenne-Primzahl, genannt M-38,

$$2^{6.972.593} - 1$$

wurde im Juni 1999 gefunden und hat 2.098.960 Stellen im Zehnersystem (das entspricht rund 77 Seiten in der FAZ).

---

<sup>9</sup>Mit CrypTool können Sie Zahlen auf folgende Weise faktorisieren: Menü **Einzelverfahren \ RSA-Kryptosystem \ Faktorisieren einer Zahl**.

In sinnvoller Zeit zerlegt CrypTool Zahlen bis 250 Bit Länge. Zahlen größer als 1024 Bit werden zur Zeit von CrypTool nicht angenommen.

Die aktuellen Faktorisierungsrekorde finden Sie in Kapitel 4.11.4.

<sup>10</sup>Auf der folgenden Seite von Landon Curt Noll werden in einer Tabelle alle Mersenne-Primzahlen samt Entdeckungsdatum und Wert in Zahlen- und Wortform aufgelistet: <http://www.isthe.com/chongo/tech/math/prime/merseenne.html>

Siehe auch: <http://www.utm.edu/>.

<sup>11</sup>Den aktuellen Status der Prüfung findet man auf der Seite: <http://www.mersenne.org/status.htm>. Hinweise, wie man Zahlen auf ihre Primalität prüfen kann, finden sich in Kapitel 3.5, Primzahltests.



## M-39 – Dezember 2001

Die 39. Mersenne-Primzahl, genannt M-39,

$$2^{13.466.917} - 1$$

wurde am 6.12.2001 bekanntgegeben: genau genommen war am 6.12.2001 die Verifikation der am 14.11.2001 von dem kanadischen Studenten Michael Cameron gefundenen Primzahl abgeschlossen. Diese Zahl hat rund 4 Millionen Stellen (genau 4.053.946 Stellen). Allein zu ihrer Darstellung

$$924947738006701322247758 \dots 1130073855470256259071$$

bräuchte man in der FAZ knapp 200 Seiten.

Inzwischen (Stand Juli 2009) wurden alle primen Exponenten kleiner als 18.000.989 getestet und nochmal geprüft<sup>12</sup>: somit können wir sicher sein, dass dies wirklich die 39. Mersenne-Primzahl ist und dass keine kleineren unentdeckten Mersenne-Primzahlen existieren (es ist üblich, die Bezeichnung M-nn erst dann zu verwenden, wenn die nn. bekannte Mersenne-Primzahl auch bewiesenermaßen die nn. Mersenne-Primzahl ist).

## GIMPS

Das GIMPS-Projekt (Great Internet Mersenne-Prime Search) wurde 1996 von George Woltman gegründet, um neue größte Mersenne-Primzahlen zu finden (<http://www.mersenne.org>). Genauere Erläuterungen zu diesem Zahlentyp finden sich unter Mersennezahlen und Mersenne-Primzahlen.

Bisher hat das GIMPS-Projekt 13 größte Mersenne-Primzahlen entdeckt, inklusive der größten bekannten Primzahl überhaupt.

Die folgende Tabelle enthält diese Mersenne Rekord-Primzahlen<sup>13,14</sup>:

Dr. Richard Crandall erfand den Transformations-Algorithmus, der im GIMPS-Programm benutzt wird. George Woltman implementierte Crandall's Algorithmus in Maschinensprache, wodurch das Primzahlenprogramm eine vorher nicht da gewesene Effizienz erhielt. Diese Arbeit führte zum GIMPS-Projekt.

Am 1. Juni 2003 wurde dem GIMPS-Server eine Zahl gemeldet, die evtl. die 40. Mersenne-Primzahl sein konnte. Diese wurde dann wie üblich überprüft, bevor sie veröffentlicht werden sollte. Leider musste der Initiator und GIMPS-Projektleiter George Woltman Mitte Juni melden, dass diese Zahl zusammengesetzt war (dies war die erste falsche positive Rückmeldung eines Clients an den Server in 7 Jahren).

Am GIMPS-Projekt beteiligen sich z.Zt. rund 130.000 freiwillige Amateure und Experten, die ihre Rechner in das von der Firma entropia organisierte „primenet“ einbinden.

---

<sup>12</sup>Siehe die Homepage des GIMPS-Projekts: [http://www.mersenne.org/report\\_milestones](http://www.mersenne.org/report_milestones).

<sup>13</sup>Eine up-to-date gehaltene Version dieser Tabelle steht im Internet unter <http://www.mersenne.org/history.htm>.

<sup>14</sup>Bei jedem neuen Rekord, der gemeldet wird, beginnen in den einschlägigen Foren die immer gleichen, oft ironischen Diskussionen: Hat diese Forschung einen tieferen Sinn? Lassen sich diese Ergebnisse für irgendwas verwenden? Die Antwort ist, dass das noch unklar ist. Aber gerade das ist bei Grundlagenforschung normal, dass man nicht sofort sieht, ob und wie es die Menschheit voranbringt.

Definition	Dezimalstellen	Wann	Wer
$2^{43.112.609} - 1$	12.978.189	23. August 2008	Edson Smith
$2^{42.643.801} - 1$	12.837.064	12. April 2009	Odd Magnar Strindmo
$2^{37.156.667} - 1$	11.185.272	6. September 2008	Hans-Michael Elvenich
$2^{32.582.657} - 1$	9.808.358	4. September 2006	Curtis Cooper/Steven Boone
$2^{30.402.457} - 1$	9.152.052	15. Dezember 2005	Curtis Cooper/Steven Boone
$2^{25.964.951} - 1$	7.816.230	18. Februar 2005	Martin Nowak
$2^{24.036.583} - 1$	7.235.733	15. Mai 2004	Josh Findley
$2^{20.996.011} - 1$	6.320.430	17. November 2003	Michael Shafer
$2^{13.466.917} - 1$	4.053.946	14. November 2001	Michael Cameron
$2^{6.972.593} - 1$	2.098.960	1. Juni 1999	Nayan Hajratwala
$2^{3.021.377} - 1$	909.526	27. Januar 1998	Roland Clarkson
$2^{2.976.221} - 1$	895.932	24. August 1997	Gordon Spence
$2^{1.398.269} - 1$	420.921	November 1996	Joel Armengaud

Tabelle 3.2: Die größten vom GIMPS-Projekt gefundenen Primzahlen (Stand Juli 2009)

### 3.4.3 Wettbewerb der Electronic Frontier Foundation (EFF)

Angefacht wird diese Suche noch zusätzlich durch einen Wettbewerb, den die Nonprofit-Organisation EFF (Electronic Frontier Foundation) mit den Mitteln eines unbekannten Spenders gestartet hat. Den Teilnehmern winken Gewinne im Gesamtwert von 500.000 USD, wenn sie die längste Primzahl finden. Dabei sucht der unbekannte Spender nicht nach dem schnellsten Rechner, sondern er will auf die Möglichkeiten des *cooperative networking* aufmerksam machen: <http://www.eff.org/awards/coop>

Der Entdecker von M-38 erhielt für die Entdeckung der ersten Primzahl mit über 1 Million Dezimalstellen von der EFF eine Prämie von 50.000 USD.

Für die von 100.000 USD von der EFF für eine Primzahl mit mehr als 10 Millionen Dezimalstellen hat sich Edson Smith qualifiziert, der im GIMPS-Projekt  $2^{43.112.609} - 1$  fand.

Nach den Preisregeln der EFF sind dann als nächste Stufe 150.000 US-Dollar für eine Primzahl mit mehr als 100 Millionen Stellen ausgelobt.

Edouard Lucas (1842-1891) hielt über 70 Jahre den Rekord der größten bekannten Primzahl, indem er nachwies, dass  $2^{127} - 1$  prim ist. Solange wird wohl kein neuer Rekord mehr Bestand haben.

## 3.5 Primzahltests<sup>15</sup>

Für die Anwendung sicherer Verschlüsselungsverfahren braucht man sehr große Primzahlen (im Bereich von  $2^{2.048}$ , das sind Zahlen im Zehnersystem mit über 600 Stellen!).

Sucht man nach den Primfaktoren, um zu entscheiden, ob eine Zahl prim ist, dauert die Suche zu lange, wenn auch der kleinste Primfaktor riesig ist. Die Zerlegung in Faktoren mittels rechnerischer systematischer Teilung oder mit dem Sieb des Eratosthenes ist mit heutigen Com-

<sup>15</sup>In dem Lernprogramm **ZT** können Sie den Fermat-Test und den Miller-Rabin-Test geführt Schritt für Schritt anwenden: Siehe Lern-Kapitel 3.2 und 3.3, Seiten 3-11/11.  
ZT können Sie in CrypTool über das Menü **Einzelverfahren** \ **Zahlentheorie interaktiv** \ **Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

putern anwendbar für Zahlen mit bis zu circa 20 Stellen im Zehnersystem. Die größte Zahl, die bisher in ihre beiden annähernd gleich großen Primfaktoren zerlegt werden konnte, hatte 200 Stellen (vgl. RSA-200 in Kapitel 4.11.4).

Ist aber etwas über die *Bauart* (spezielle Struktur) der fraglichen Zahl bekannt, gibt es sehr hochentwickelte Verfahren, die deutlich schneller sind. Diese Verfahren beantworten nur die Primalitätseigenschaft einer Zahl, können aber nicht die Primfaktoren sehr großer zusammengesetzter Zahlen bestimmen.

Fermat<sup>16</sup> hatte im 17. Jahrhundert an Mersenne geschrieben, dass er vermute, dass alle Zahlen der Form

$$f(n) = 2^{2^n} + 1$$

für alle ganzen Zahlen  $n \geq 0$  prim seien (siehe unten, Kapitel 3.6.4).

Schon im 19. Jahrhundert wusste man, dass die 29-stellige Zahl

$$f(7) = 2^{2^7} + 1$$

keine Primzahl ist. Aber erst 1970 fanden Morrison/Billhart ihre Zerlegung.

$$\begin{aligned} f(7) &= 340.282.366.920.938.463.463.374.607.431.768.211.457 \\ &= 59.649.589.127.497.217 \cdot 5.704.689.200.685.129.054.721 \end{aligned}$$

Auch wenn sich Fermat bei seiner Vermutung irrte, so stammt in diesem Zusammenhang von ihm doch ein sehr wichtiger Satz: Der (kleine) Fermatsche Satz, den Fermat im Jahr 1640 aufstellte, ist der Ausgangspunkt vieler schneller Primzahltests (siehe Kap. 4.8.3).

**Satz 3.5.1** („kleiner“ Fermat). *Sei  $p$  eine Primzahl und  $a$  eine beliebige ganze Zahl, dann gilt für alle  $a$*

$$a^p \equiv a \pmod{p}.$$

*Eine alternative Formulierung lautet:*

*Sei  $p$  eine Primzahl und  $a$  eine beliebige ganze Zahl, die kein Vielfaches von  $p$  ist (also  $a \not\equiv 0 \pmod{p}$ ), dann gilt  $a^{p-1} \equiv 1 \pmod{p}$ .*

Wer mit dem Rechnen mit Resten (Modulo-Rechnung) nicht so vertraut ist, möge den Satz einfach so hinnehmen oder erst Kapitel 4 „Einführung in die elementare Zahlentheorie mit Beispielen“ lesen. Wichtig ist, dass aus diesem Satz folgt, dass wenn diese Gleichheit für irgendein ganzes  $a$  nicht erfüllt ist, dann ist  $p$  keine Primzahl! Die Tests lassen sich (zum Beispiel für die erste Formulierung) leicht mit der *Testbasis*  $a = 2$  durchführen.

Damit hat man ein Kriterium für Nicht-Primzahlen, also einen negativen Test, aber noch keinen Beweis, dass eine Zahl  $a$  prim ist. Leider gilt die Umkehrung zum Fermatschen Satz nicht, sonst hätten wir einen einfachen Beweis für die Primzahleigenschaft (man sagt auch, man hätte dann ein einfaches Primzahlkriterium).

## Pseudoprimzahlen

Zahlen  $n$ , die die Eigenschaft

$$2^n \equiv 2 \pmod{n}$$

---

<sup>16</sup>Pierre de Fermat, französischer Mathematiker, 17.8.1601 – 12.1.1665.

erfüllen, aber nicht prim sind, bezeichnet man als *Pseudoprimzahlen* (der Exponent  $n$  ist also keine Primzahl). Die erste Pseudoprimzahl ist

$$341 = 11 \cdot 31.$$

## Carmichaelzahlen

Es gibt Pseudoprimzahlen  $n$ , die den Fermat-Test

$$a^{n-1} \equiv 1 \pmod{n}$$

mit allen Basen  $a$ , die teilerfremd zu  $n$  sind [ $\gcd(a, n) = 1$ ], bestehen, obwohl die zu testenden Zahlen  $n$  nicht prim sind: Diese Zahlen heißen *Carmichaelzahlen*. Die erste ist

$$561 = 3 \cdot 11 \cdot 17.$$

### Beispiel: Die zu testende Zahl sei 561.

Da  $561 = 3 \cdot 11 \cdot 17$  ist, ergibt sich:

Die Testbedingung  $a^{560} \pmod{561} = 1$

ist erfüllt für  $a = 2, 4, 5, 7, \dots$ ,

aber nicht für  $a = 3, 6, 9, 11, 12, 15, 17, 18, 21, 22, \dots$ .

D.h. die Testbedingung muss nicht erfüllt sein, wenn die Basis ein Vielfaches von 3, von 11 oder von 17 ist.

Der Test angewandt auf  $a = 3$  ergibt:  $3^{560} \pmod{561} = 375$ .

Der Test angewandt auf  $a = 5$  ergibt:  $5^{560} \pmod{561} = 1$ .

## Starke Pseudoprimzahlen

Ein stärkerer Test stammt von Miller/Rabin<sup>17</sup>: er wird nur von sogenannten *starken Pseudoprimzahlen* bestanden. Wiederum gibt es starke Pseudoprimzahlen, die keine Primzahlen sind, aber das passiert deutlich seltener als bei den (einfachen) Pseudoprimzahlen oder bei den Carmichaelzahlen. Die kleinste starke Pseudoprimzahl zur Basis 2 ist

$$15.841 = 7 \cdot 31 \cdot 73.$$

Testet man alle 4 Basen 2, 3, 5 und 7, so findet man bis  $25 \cdot 10^9$  nur eine starke Pseudoprimzahl, also eine Zahl, die den Test besteht und doch keine Primzahl ist.

Weiterführende Mathematik hinter dem Rabin-Test gibt dann die Wahrscheinlichkeit an, mit der die untersuchte Zahl prim ist (solche Wahrscheinlichkeiten liegen heutzutage bei circa  $10^{-60}$ ).

Ausführliche Beschreibungen zu Tests, um herauszufinden, ob eine Zahl prim ist, finden sich zum Beispiel unter:

<http://www.utm.edu/research/primes/merenne.shtml>

<http://www.utm.edu/research/primes/prove/index.html>

---

<sup>17</sup>1976 veröffentlichte Prof. Rabin einen effizienten probabilistischen Primzahltest, der auf einem zahlentheoretischen Ergebnis von Prof. Miller aus der Jahr davor basierte.

Prof. Miller arbeitete an der Carnegie-Mellon Universität, School of Computer Science. Prof. Rabin, geboren 1931, arbeitete an der Harvard und Hebrew Universität.

## 3.6 Übersicht Spezial-Zahlentypen und die Suche nach einer Formel für Primzahlen

Derzeit sind keine brauchbaren, offenen (also nicht rekursiven) Formeln bekannt, die nur Primzahlen liefern (rekursiv bedeutet, dass zur Berechnung der Funktion auf dieselbe Funktion in Abhängigkeit einer kleineren Variablen zugegriffen wird). Die Mathematiker wären schon zufrieden, wenn sie eine Formel fänden, die wohl Lücken lässt (also nicht alle Primzahlen liefert), aber sonst keine zusammengesetzten Zahlen (Nicht-Primzahlen) liefert.

Optimal wäre, man würde für das Argument  $n$  sofort die  $n$ -te Primzahl bekommen, also für  $f(8) = 19$  oder für  $f(52) = 239$ .

Ideen dazu finden sich in

[http://www.utm.edu/research/primenes/notes/faq/p\\_n.html](http://www.utm.edu/research/primenes/notes/faq/p_n.html).

Die Tabelle unter 3.10 enthält die exakten Werte für die  $n$ -ten Primzahlen für ausgewählte  $n$ .

Für „Primzahlformeln“ werden meist ganz spezielle Zahlentypen benutzt. Die folgende Aufzählung enthält die verbreitetsten Ansätze für „Primzahlformeln“, und welche Kenntnisse wir über sehr große Folgenglieder haben: Konnte die Primalität bewiesen werden? Wenn es zusammengesetzte Zahlen sind, konnten die Primfaktoren bestimmt werden?

### 3.6.1 Mersennezahlen $f(n) = 2^n - 1$ für $n$ prim

Wie oben gesehen, liefert diese Formel wohl relativ viele große Primzahlen, aber es kommt – wie für  $n = 11$  [ $f(n) = 2.047$ ] – immer wieder vor, dass das Ergebnis auch bei primen Exponenten nicht prim ist.

Heute kennt man alle Mersenne-Primzahlen mit bis zu ca. 4.000.000 Dezimalstellen (M-39):

<http://perso.wanadoo.fr/yves.gallot/primenes/index.html>

### 3.6.2 Verallgemeinerte Mersennezahlen $f(k, n) = k \cdot 2^n \pm 1$ für $n$ prim und $k$ kleine Primzahl / Proth-Zahlen<sup>18</sup>

Diese 1. Verallgemeinerung der Mersennezahlen erzeugt die sogenannten Proth-Zahlen. Dafür gibt es (für kleine  $k$ ) ebenfalls sehr schnelle Primzahltests (vgl. [Knuth1981]).

Praktisch ausführen lässt sich das zum Beispiel mit der Software Proths von Yves Gallot:

<http://www.prothsearch.net/index.html>.

### 3.6.3 Verallgemeinerte Mersennezahlen $f(b, n) = b^n \pm 1$ / Cunningham-Projekt

Dies ist eine 2. mögliche Verallgemeinerung der Mersennezahlen. Im **Cunningham-Projekt** werden die Faktoren aller zusammengesetzten Zahlen bestimmt, die sich in folgender Weise bilden:

$$f(b, n) = b^n \pm 1 \quad \text{für } b = 2, 3, 5, 6, 7, 10, 11, 12$$

<sup>18</sup>Diese wurden nach dem französischen Landwirt François Proth (1852-1879) benannt. Berühmter noch als die Proth-Primzahlen dürfte das eng damit zusammenhängende Sierpinski-Problem sein, nämlich Zahlen  $k$  zu finden, so dass  $k \cdot 2^n + 1$  zusammengesetzt ist für alle  $n > 0$ . Siehe Tab. 3.1.

( $b$  ist ungleich der Vielfachen von schon benutzten Basen wie 4, 8, 9).

Details hierzu finden sich unter:

<http://www.cerias.purdue.edu/homes/ssw/cun>

### 3.6.4 Fermatzahlen<sup>19</sup> $f(n) = 2^{2^n} + 1$

Wie oben in Kapitel 3.5 erwähnt, schrieb Fermat an Mersenne, dass er vermutet, dass alle Zahlen dieser Form prim seien. Diese Vermutung wurde jedoch von Euler (1732) widerlegt. Es gilt  $641|f(5)$ <sup>20</sup>.

$f(0) = 2^{2^0} + 1 = 2^1 + 1$	$= 3$	$\mapsto$ prim
$f(1) = 2^{2^1} + 1 = 2^2 + 1$	$= 5$	$\mapsto$ prim
$f(2) = 2^{2^2} + 1 = 2^4 + 1$	$= 17$	$\mapsto$ prim
$f(3) = 2^{2^3} + 1 = 2^8 + 1$	$= 257$	$\mapsto$ prim
$f(4) = 2^{2^4} + 1 = 2^{16} + 1$	$= 65.537$	$\mapsto$ prim
$f(5) = 2^{2^5} + 1 = 2^{32} + 1$	$= 4.294.967.297 = 641 \cdot 6.700.417$	$\mapsto$ NICHT prim!
$f(6) = 2^{2^6} + 1 = 2^{64} + 1$	$= 18.446.744.073.709.551.617$	
	$= 274.177 \cdot 67.280.421.310.721$	$\mapsto$ NICHT prim!
$f(7) = 2^{2^7} + 1 = 2^{128} + 1$	$= (\text{siehe Seite 61})$	$\mapsto$ NICHT prim!

Innerhalb des Projektes “Distributed Search for Fermat Number Dividers”, das von Leonid Durman angeboten wird, gibt es ebenfalls Fortschritte beim Finden von neuen Primzahl-Riesen (<http://www.fermatsearch.org/> – diese Webseite hat Verknüpfungen zu Seiten in russisch, italienisch und deutsch).

Die entdeckten Faktoren können sowohl zusammengesetzte natürliche als auch prime natürliche Zahlen sein.

Am 22. Februar 2003 entdeckte John Cosgrave

- die größte bis dahin bekannte zusammengesetzte Fermatzahl und
- die größte bis dahin bekannte prime nicht-einfache Mersennezahl mit 645.817 Dezimalstellen.

Die Fermatzahl

$$f(2.145.351) = 2^{(2^{2.145.351})} + 1$$

ist teilbar durch die Primzahl

$$p = 3 * 2^{2.145.353} + 1$$

Diese Primzahl  $p$  war damals die größte bis dahin bekannte prime verallgemeinerte Mersennezahl und die 5.-größte damals bekannte Primzahl überhaupt.

<sup>19</sup>Die Fermatschen Primzahlen spielen unter anderem eine wichtige Rolle in der Kreisteilung. Wie Gauss bewiesen hat, ist das reguläre  $p$ -Eck für eine Primzahl  $p > 2$  dann und nur dann mit Zirkel und Lineal konstruierbar, wenn  $p$  eine Fermatsche Primzahl ist.

<sup>20</sup>Erstaunlicherweise kann man mit Hilfe des Satzes von Fermat diese Zahl leicht finden (siehe z.B. [Scheid1994, S. 176])

Zu diesem Erfolg trugen bei: NewPGen von Paul Jobling's, PRP von George Woltman's, Proth von Yves Gallot's Programm und die Proth-Gallot-Gruppe am St. Patrick's College, Dublin.

Weitere Details finden sich unter

[http://www.fermatsearch.org/history/cosgrave\\_record.htm/](http://www.fermatsearch.org/history/cosgrave_record.htm/)

### 3.6.5 Verallgemeinerte Fermatzahlen<sup>21</sup> $f(b, n) = b^{2^n} + 1$

Verallgemeinerte Fermatzahlen kommen häufiger vor als Mersennezahlen gleicher Größe, so dass wahrscheinlich noch viele gefunden werden können, die die großen Lücken zwischen den Mersenne-Primzahlen verkleinern. Fortschritte in der Zahlentheorie haben es ermöglicht, dass Zahlen, deren Repräsentation nicht auf eine Basis von 2 beschränkt ist, nun mit fast der gleichen Geschwindigkeit wie Mersennezahlen getestet werden können.

Yves Gallot schrieb das Programm Proth.exe zur Untersuchung verallgemeinerter Fermatzahlen.

Mit diesem Programm fand Michael Angel am 16. Februar 2003 eine prime verallgemeinerte Fermatzahl mit 628.808 Dezimalstellen, die zum damaligen Zeitpunkt zur 5.-größten bis dahin bekannten Primzahl wurde:

$$b^{2^{17}} + 1 = 62.722^{131.072} + 1.$$

Weitere Details finden sich unter

<http://primes.utm.edu/top20/page.php?id=12>

### 3.6.6 Carmichaelzahlen

Wie oben in Kapitel 3.5 erwähnt, sind nicht alle Carmichaelzahlen prim.

### 3.6.7 Pseudoprimzahlen

Siehe oben in Kapitel 3.5.

### 3.6.8 Starke Pseudoprimzahlen

Siehe oben in Kapitel 3.5.

### 3.6.9 Idee aufgrund von Euklids Beweis $p_1 \cdot p_2 \cdots p_n + 1$

Diese Idee entstammt Euklids Beweis, dass es unendlich viele Primzahlen gibt.

$2 \cdot 3 + 1$	$= 7$	$\mapsto$ prim
$2 \cdot 3 \cdot 5 + 1$	$= 31$	$\mapsto$ prim
$2 \cdot 3 \cdot 5 \cdot 7 + 1$	$= 211$	$\mapsto$ prim
$2 \cdot 3 \cdots 11 + 1$	$= 2311$	$\mapsto$ prim
$2 \cdot 3 \cdots 13 + 1$	$= 59 \cdot 509$	$\mapsto$ NICHT prim!
$2 \cdot 3 \cdots 17 + 1$	$= 19 \cdot 97 \cdot 277$	$\mapsto$ NICHT prim!

<sup>21</sup>Hier ist die Basis b nicht notwendigerweise 2. Noch allgemeiner wäre:  $f(b, c, n) = b^{c^n} \pm 1$

### 3.6.10 Wie zuvor, nur $-1$ statt $+1$ : $p_1 \cdot p_2 \cdots p_n - 1$

$$\begin{array}{lll}
2 \cdot 3 - 1 & = 5 & \mapsto \text{prim} \\
2 \cdot 3 \cdot 5 - 1 & = 29 & \mapsto \text{prim} \\
2 \cdot 3 \cdots 7 - 1 & = 11 \cdot 19 & \mapsto \text{NICHT prim!} \\
2 \cdot 3 \cdots 11 - 1 & = 2309 & \mapsto \text{prim} \\
2 \cdot 3 \cdots 13 - 1 & = 30029 & \mapsto \text{prim} \\
2 \cdot 3 \cdots 17 - 1 & = 61 \cdot 8369 & \mapsto \text{NICHT prim!}
\end{array}$$

### 3.6.11 Euklidzahlen $e_n = e_0 \cdot e_1 \cdots e_{n-1} + 1$ mit $n \geq 1$ und $e_0 := 1$

$e_{n-1}$  ist nicht die  $(n-1)$ -te Primzahl, sondern die zuvor hier gefundene Zahl. Diese Formel ist leider nicht offen, sondern rekursiv. Die Folge startet mit

$$\begin{array}{lll}
e_1 = 1 + 1 & = 2 & \mapsto \text{prim} \\
e_2 = e_1 + 1 & = 3 & \mapsto \text{prim} \\
e_3 = e_1 \cdot e_2 + 1 & = 7 & \mapsto \text{prim} \\
e_4 = e_1 \cdot e_2 \cdot e_3 + 1 & = 43 & \mapsto \text{prim} \\
e_5 = e_1 \cdot e_2 \cdots e_4 + 1 & = 13 \cdot 139 & \mapsto \text{NICHT prim!} \\
e_6 = e_1 \cdot e_2 \cdots e_5 + 1 & = 3.263.443 & \mapsto \text{prim} \\
e_7 = e_1 \cdot e_2 \cdots e_6 + 1 & = 547 \cdot 607 \cdot 1.033 \cdot 31.051 & \mapsto \text{NICHT prim!} \\
e_8 = e_1 \cdot e_2 \cdots e_7 + 1 & = 29.881 \cdot 67.003 \cdot 9.119.521 \cdot 6.212.157.481 & \mapsto \text{NICHT prim!}
\end{array}$$

Auch  $e_9, \dots, e_{17}$  sind zusammengesetzt, so dass dies auch keine brauchbare Primzahlformel ist.

#### Bemerkung:

Das Besondere an diesen Zahlen ist, dass sie jeweils paarweise keinen gemeinsamen Teiler außer 1 haben<sup>22</sup>, sie sind also *relativ zueinander prim*.

<sup>22</sup>Dies kann leicht gezeigt werden mit Hilfe der Rechenregel für den *größten gemeinsamen Teiler*  $ggT$  mit  $ggT(a, b) = ggT(b - \lfloor b/a \rfloor \cdot a, a)$ .

Es gilt für  $i < j$ :

$ggT(e_i, e_j) \leq ggT(e_1 \cdots e_i \cdots e_{j-1}, e_j) = ggT(e_j - e_1 \cdots e_i \cdots e_{j-1}, e_1 \cdots e_i \cdots e_{j-1}) = ggT(1, e_1 \cdots e_i \cdots e_{j-1}) = 1$ .

Siehe Seite 149.



### 3.6.12 $f(n) = n^2 + n + 41$

Diese Folge hat einen sehr *erfolgversprechenden* Anfang, aber das ist noch lange kein Beweis.

$f(0) = 41$	$\mapsto$	prim
$f(1) = 43$	$\mapsto$	prim
$f(2) = 47$	$\mapsto$	prim
$f(3) = 53$	$\mapsto$	prim
$f(4) = 61$	$\mapsto$	prim
$f(5) = 71$	$\mapsto$	prim
$f(6) = 83$	$\mapsto$	prim
$f(7) = 97$	$\mapsto$	prim
$\vdots$		
$f(33) = 1.163$	$\mapsto$	prim
$f(34) = 1.231$	$\mapsto$	prim
$f(35) = 1.301$	$\mapsto$	prim
$f(36) = 1.373$	$\mapsto$	prim
$f(37) = 1.447$	$\mapsto$	prim
$f(38) = 1.523$	$\mapsto$	prim
$f(39) = 1.601$	$\mapsto$	prim
$f(40) = 1681$	$= 41 \cdot 41$	$\mapsto$ NICHT prim!
$f(41) = 1763$	$= 41 \cdot 43$	$\mapsto$ NICHT prim!

Die ersten 40 Werte sind Primzahlen (diese haben die auffallende Regelmäßigkeit, dass ihr Abstand beginnend mit dem Abstand 2 jeweils um 2 wächst), aber der 41. und der 42. Wert sind keine Primzahlen. Dass  $f(41)$  keine Primzahl sein kann, lässt sich leicht überlegen:  $f(41) = 41^2 + 41 + 41 = 41(41 + 1 + 1) = 41 \cdot 43$ .

### 3.6.13 $f(n) = n^2 - 79 \cdot n + 1.601$

Diese Funktion liefert für die Werte  $n = 0$  bis  $n = 79$  stets Primzahlwerte<sup>23</sup>. Leider ergibt  $f(80) = 1.681 = 11 \cdot 151$  keine Primzahl. Bis heute kennt man keine Funktion, die mehr aufeinanderfolgende Primzahlen annimmt. Andererseits kommt jede Primzahl doppelt vor (erst in der absteigenden, dann in der aufsteigenden Folge), so dass sie insgesamt genau 40 verschiedene Primzahlwerte in Folge liefert (Es sind dieselben wie die, die die Funktion aus Kapitel 3.6.12 liefert)<sup>24</sup>.

<sup>23</sup>In Kapitel 3.14 “Anhang: Beispiele mit Sage” finden Sie den Quellcode zur Berechnung der Tabelle mit Sage.

<sup>24</sup>Ein weiteres quadratisches Polynom, das dieselben Primzahlen liefert, ist:  $f(n) = n^2 - 9 \cdot n + 61$ .

Unter den ersten 1000 Folgengliedern dieser Funktion sind über 50% prim (vgl. Kapitel 3.14 “Anhang: Beispiele mit Sage”).

$f(0) = 1.601 \mapsto \text{prim}$	$f(26) = 223 \mapsto \text{prim}$
$f(1) = 1.523 \mapsto \text{prim}$	$f(27) = 197 \mapsto \text{prim}$
$f(2) = 1.447 \mapsto \text{prim}$	$f(28) = 173 \mapsto \text{prim}$
$f(3) = 1.373 \mapsto \text{prim}$	$f(29) = 151 \mapsto \text{prim}$
$f(4) = 1.301 \mapsto \text{prim}$	$f(30) = 131 \mapsto \text{prim}$
$f(5) = 1.231 \mapsto \text{prim}$	$f(31) = 113 \mapsto \text{prim}$
$f(6) = 1.163 \mapsto \text{prim}$	$f(32) = 97 \mapsto \text{prim}$
$f(7) = 1.097 \mapsto \text{prim}$	$f(33) = 83 \mapsto \text{prim}$
$f(8) = 1.033 \mapsto \text{prim}$	$f(34) = 71 \mapsto \text{prim}$
$f(9) = 971 \mapsto \text{prim}$	$f(35) = 61 \mapsto \text{prim}$
$f(10) = 911 \mapsto \text{prim}$	$f(36) = 53 \mapsto \text{prim}$
$f(11) = 853 \mapsto \text{prim}$	$f(37) = 47 \mapsto \text{prim}$
$f(12) = 797 \mapsto \text{prim}$	$f(38) = 43 \mapsto \text{prim}$
$f(13) = 743 \mapsto \text{prim}$	$f(39) = 41 \mapsto \text{prim}$
$f(14) = 691 \mapsto \text{prim}$	$f(40) = 41 \mapsto \text{prim}$
$f(15) = 641 \mapsto \text{prim}$	$f(41) = 43 \mapsto \text{prim}$
$f(16) = 593 \mapsto \text{prim}$	$f(42) = 47 \mapsto \text{prim}$
$f(17) = 547 \mapsto \text{prim}$	$f(43) = 53 \mapsto \text{prim}$
$f(18) = 503 \mapsto \text{prim}$	$\dots$
$f(19) = 461 \mapsto \text{prim}$	$f(77) = 1.447 \mapsto \text{prim}$
$f(20) = 421 \mapsto \text{prim}$	$f(78) = 1.523 \mapsto \text{prim}$
$f(21) = 383 \mapsto \text{prim}$	$f(79) = 1.601 \mapsto \text{prim}$
$f(22) = 347 \mapsto \text{prim}$	$f(80) = 41 \cdot 41 \mapsto \text{NICHT prim!}$
$f(21) = 383 \mapsto \text{prim}$	$f(81) = 41 \cdot 43 \mapsto \text{NICHT prim!}$
$f(22) = 347 \mapsto \text{prim}$	$f(82) = 1.847 \mapsto \text{prim}$
$f(23) = 313 \mapsto \text{prim}$	$f(83) = 1.933 \mapsto \text{prim}$
$f(24) = 281 \mapsto \text{prim}$	$f(84) = 43 \cdot 47 \mapsto \text{NICHT prim!}$
$f(25) = 251 \mapsto \text{prim}$	

### 3.6.14 Polynomfunktionen $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$ ( $a_i$ aus $\mathbb{Z}$ , $n \geq 1$ )

Es existiert kein solches Polynom, das für alle  $x$  aus  $\mathbb{Z}$  ausschließlich Primzahlwerte annimmt. Zum Beweis sei auf [Padberg1996, S. 83 f.], verwiesen, wo sich auch weitere Details zu Primzahlformeln finden.

Damit ist es hoffnungslos, weiter nach Formeln (Funktionen) wie in Kapitel 3.6.12 oder Kapitel 3.6.13 zu suchen.

### 3.6.15 Vermutung von Catalan<sup>25</sup>

Catalan äußerte die Vermutung, dass  $C_4$  eine Primzahl ist:

$$\begin{aligned}C_0 &= 2, \\C_1 &= 2^{C_0} - 1, \\C_2 &= 2^{C_1} - 1, \\C_3 &= 2^{C_2} - 1, \\C_4 &= 2^{C_3} - 1, \dots\end{aligned}$$

(siehe <http://www.utm.edu/research/primes/merenne.shtml> unter Conjectures and Unsolved Problems).

Diese Folge ist ebenfalls rekursiv definiert und wächst sehr schnell. Besteht sie nur aus Primzahlen?

$C(0) = 2$	$\mapsto$	prim
$C(1) = 2^2 - 1 = 3$	$\mapsto$	prim
$C(2) = 2^3 - 1 = 7$	$\mapsto$	prim
$C(3) = 2^7 - 1 = 127$	$\mapsto$	prim
$C(4) = 2^{127} - 1 = 170.141.183.460.469.231.731.687.303.715.884.105.727$	$\mapsto$	prim

Ob  $C_5$  bzw. alle höheren Elemente prim sind, ist (noch) nicht bekannt, aber auch nicht wahrscheinlich. Bewiesen ist jedenfalls nicht, dass diese Formel nur Primzahlen liefert.

## 3.7 Dichte und Verteilung der Primzahlen

Wie Euklid herausfand, gibt es unendlich viele Primzahlen. Einige unendliche Mengen sind aber *dichter* als andere.

Innerhalb der natürlichen Zahlen gibt es unendlich viele gerade, ungerade und quadratische Zahlen. Wie man die „Dichte“ zweier unendlicher Mengen vergleicht, soll kurz anhand der geraden und quadratischen Zahlen erläutert werden.

Nach folgenden Gesichtspunkten ist die Menge der geraden Zahlen dichter als die Menge der quadratischen Zahlen:<sup>26</sup>

- die Größe des  $n$ -ten Elements:  
Das  $n$ -te Element der geraden Zahlen ist  $2n$ ; das  $n$ -te Element der Quadratzahlen ist  $n^2$ . Weil für alle  $n > 2$  gilt:  $2n < n^2$ , kommt die  $n$ -te gerade Zahl viel früher als die  $n$ -te quadratische Zahl.
- die Anzahlen der Werte, die kleiner oder gleich einem bestimmten *Dachwert*  $x$  aus  $\mathbb{R}$  sind:  
Es gibt  $\lfloor x/2 \rfloor$  solcher gerader Zahlen und  $\lfloor \sqrt{x} \rfloor$  Quadratzahlen. Da für alle  $x > 6$  gilt, dass der Wert  $\lfloor x/2 \rfloor$  größer ist als die größte ganze Zahl kleiner oder gleich der Quadratwurzel aus  $x$ , sind die geraden Zahlen dichter verteilt.

<sup>25</sup>Eugene Charles Catalan, belgischer Mathematiker, 30.5.1814–14.2.1894.

Nach ihm sind auch die sogenannten *Catalanzahlen*  $A(n) = (1/(n+1)) * (2n)!/(n!)^2$   
 $= 1, 2, 5, 14, 42, 132, 429, 1.430, 4.862, 16.796, 58.786, 208.012, 742.900, 2.674.440, 9.694.845, \dots$  benannt.

<sup>26</sup>Während in der Umgangssprache oft gesagt wird, es „gibt mehr“ gerade als quadratische Zahlen, sagen Mathematiker, dass es von beiden unendlich viele gibt, dass ihre Mengen äquivalent zu  $\mathbb{N}$  sind (also beide unendlich und abzählbar, d.h. man kann für jede gerade Zahl und für jede quadratische Zahl eine natürliche Zahl angeben), dass aber die Menge der geraden Zahlen dichter ist als die der quadratischen Zahlen. Mathematiker haben für die Mächtigkeit von Mengen also sehr präzise Formulierungen gefunden.

## Der Wert der $n$ -ten Primzahl $P(n)$

**Satz 3.7.1.** Für große  $n$  gilt: Der Wert der  $n$ -ten Primzahl  $P(n)$  ist asymptotisch zu  $n \cdot \ln(n)$ , d.h. der Grenzwert des Verhältnisses  $P(n)/(n \cdot \ln n)$  ist gleich 1, wenn  $n$  gegen unendlich geht.

Es gilt für  $n \geq 5$ , dass  $P(n)$  zwischen  $2n$  und  $n^2$  liegt. Es gibt also weniger Primzahlen als gerade natürliche Zahlen, aber es gibt mehr Primzahlen als Quadratzahlen<sup>27</sup>.

## Die Anzahl der Primzahlen $PI(x)$

Ähnlich wird die Anzahl der Primzahlen  $PI(x)$  definiert, die den Dachwert  $x$  nicht übersteigen:

**Satz 3.7.2.**  $PI(x)$  ist asymptotisch zu  $x/\ln(x)$ .

Dies ist der **Primzahlsatz** (prime number theorem). Er wurde von Legendre<sup>28</sup> und Gauss<sup>29</sup> aufgestellt und erst über 100 Jahre später bewiesen.

Die Übersicht unter 3.9 zeigt die Anzahl von Primzahlen in verschiedenen Intervallen.

Diese Formeln, die nur für  $n$  gegen unendlich gelten, können durch präzisere Formeln ersetzt werden. Für  $x \geq 67$  gilt:

$$\ln(x) - 1,5 < x/PI(x) < \ln(x) - 0,5$$

Im Bewusstsein, dass  $PI(x) = x/\ln x$  nur für sehr große  $x$  ( $x$  gegen unendlich) gilt, kann man folgende Übersicht erstellen:

$x$	$\ln(x)$	$x/\ln(x)$	$PI(x)$ (gezählt)	$PI(x)/(x/\ln(x))$
$10^3$	6,908	144	168	1,160
$10^6$	13,816	72.386	78.498	1,085
$10^9$	20,723	48.254.942	50.847.534	1,054

Für eine Binärzahl<sup>30</sup>  $x$  der Länge 250 Bit ( $2^{250}$  ist ungefähr  $= 1,809251 \cdot 10^{75}$ ) gilt:

$$PI(x) = 2^{250}/(250 \cdot \ln 2) \text{ ist ungefähr } = 2^{250}/173,28677 = 1,045810 \cdot 10^{73}.$$

Es ist also zu erwarten, dass sich innerhalb der Zahlen der Bitlänge kleiner als 250 ungefähr  $10^{73}$  Primzahlen befinden (ein beruhigendes Ergebnis?!).

Man kann das auch so formulieren: Betrachtet man eine zufällige natürliche Zahl  $n$ , so sind die Chancen, dass diese Zahl prim ist, circa  $1/\ln(n)$ . Nehmen wir zum Beispiel Zahlen in der Gegend von  $10^{16}$ , so müssen wir ungefähr (durchschnittlich)  $16 \cdot \ln 10 = 36,8$  Zahlen betrachten, bis wir eine Primzahl finden. Ein genaue Untersuchung zeigt: Zwischen  $10^{16} - 370$  und  $10^{16} - 1$  gibt es 10 Primzahlen.

Unter der Überschrift *How Many Primes Are There* finden sich unter

<http://www.utm.edu/research/primes/howmany.shtml>

viele weitere Details.

$PI(x)$  lässt sich leicht per

<sup>27</sup>Vergleiche auch Tabelle 3.10.

<sup>28</sup>Adrien-Marie Legendre, französischer Mathematiker, 18.9.1752–10.1.1833.

<sup>29</sup>Carl Friedrich Gauss, deutscher Mathematiker und Astronom, 30.4.1777–23.2.1855.

<sup>30</sup>Eine Zahl im Zweiersystem besteht nur aus den Ziffern 0 und 1.

<http://www.math.Princeton.EDU/~arbooker/nthprime.html>

bestimmen.

Die **Verteilung** der Primzahlen<sup>31</sup> weist viele Unregelmäßigkeiten auf, für die bis heute kein „System“ gefunden wurde: Einerseits liegen viele eng benachbart wie 2 und 3, 11 und 13, 809 und 811, andererseits tauchen auch längere Primzahlücken auf. So liegen zum Beispiel zwischen 113 und 127, 293 und 307, 317 und 331, 523 und 541, 773 und 787, 839 und 853 sowie zwischen 887 und 907 keine Primzahlen.

Details siehe:

<http://www.utm.edu/research/primenumbers/notes/gaps.html>

Gerade dies macht einen Teil des Ehrgeizes der Mathematiker aus, ihre Geheimnisse herauszufinden.

### Sieb des Eratosthenes

Ein einfacher Weg, alle  $PI(x)$  Primzahlen kleiner oder gleich  $x$  zu berechnen, ist das Sieb des Eratosthenes. Er fand schon im 3. Jahrhundert vor Christus einen sehr einfach automatisierbaren Weg, das herauszufinden. Zuerst werden ab 2 alle Zahlen bis  $x$  aufgeschrieben, die 2 umkreist und dann streicht man alle Vielfachen von 2. Anschließend umkreist man die kleinste noch nicht umkreiste oder gestrichene Zahl (3), streicht wieder alle ihre Vielfachen, usw. Durchzuführen braucht man das nur bis zu der größten Zahl, deren Quadrat kleiner oder gleich  $x$  ist.<sup>32</sup>

Abgesehen von 2 sind Primzahlen nie gerade. Abgesehen von 2 und 5 haben Primzahlen nie die Endziffern 2, 5 oder 0. Also braucht man sowieso nur Zahlen mit den Endziffern 1, 3, 7, 9 zu betrachten (es gibt unendlich viele Primzahlen mit jeder dieser letzten Ziffern; vergleiche [Tietze1973, Bd. 1, S. 137]).

Inzwischen findet man im Internet auch viele fertige Programme, oft mit komplettem Quellcode, so dass man auch selbst mit großen Zahlen experimentieren kann (vergleiche Kap. 3.6). Ebenfalls zugänglich sind große Datenbanken, die entweder viele Primzahlen oder die Zerlegung in Primfaktoren vieler zusammengesetzter Zahlen enthalten.

### Weitere interessante Themen rund um Primzahlen

In diesem Kapitel 3 wurden weitere, eher zahlentheoretische Themen wie Teilbarkeitsregeln, Modulo-Rechnung, modulare Inverse, modulare Potenzen und Wurzeln, chinesischer Restesatz, Eulersche Phi-Funktion und perfekte Zahlen nicht betrachtet. Auf einige dieser Themen geht das **nächste Kapitel** (Kapitel 4) ein.

---

<sup>31</sup>Einige Visualisierungen (Plots) zur Menge von Primzahlen in verschiedenen Zahlendimensionen finden Sie in Kapitel 3.13 „Anhang: Visualisierung der Menge der Primzahlen in hohen Bereichen“.

<sup>32</sup>Mit dem Lernprogramm **ZT** können Sie für beliebige eigene Wertemengen das Sieb des Eratosthenes rechnergestützt und geführt Schritt für Schritt anwenden: Siehe Lern-Kapitel 1.2, Seite 6/21 und 7/21.

ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

## 3.8 Anmerkungen zu Primzahlen

Die folgenden Anmerkungen listen einzelne interessante Sätze, Vermutungen und Fragestellungen zu Primzahlen auf, aber auch Kurioses und Übersichten.

### 3.8.1 Bewiesene Aussagen / Sätze zu Primzahlen

- Zu jeder Zahl  $n$  aus  $\mathbf{N}$  gibt es  $n$  aufeinanderfolgende natürliche Zahlen, die keine Primzahlen sind. Ein Beweis findet sich in [Padberg1996, S. 79].
- Paul Erdős<sup>33</sup> bewies: Zwischen jeder beliebigen Zahl ungleich 1 und ihrem Doppelten gibt es mindestens eine Primzahl. Er bewies das Theorem nicht als erster, aber auf einfachere Weise als andere vor ihm.
- Es existiert eine reelle Zahl  $a$ , so dass die Funktion  $f : \mathbf{N} \rightarrow \mathbb{Z}$  mit  $n \mapsto \lfloor a^{3^n} \rfloor$  für alle  $n$  nur Primzahlenwerte annimmt<sup>34</sup> (siehe [Padberg1996, S. 82]). Leider macht die Bestimmung von  $a$  Probleme (siehe Kapitel 3.8.4).

- Es gibt arithmetische Primzahlfolgen beliebig großer Länge.<sup>35,36</sup>

Die 1923 von dem berühmten englischen Mathematiker Godfrey Harold Hardy<sup>37</sup> aufgestellte Vermutung, dass es arithmetische Folgen beliebiger Länge gibt, die nur aus Primzahlen bestehen, wurde 2004 von zwei jungen amerikanischen Mathematikern bewiesen.

Jedes Schulkind lernt im Mathematikunterricht irgendwann einmal die arithmetischen Zahlenfolgen kennen. Das sind Aneinanderreihungen von Zahlen, bei denen die Abstände zwischen je zwei aufeinander folgenden Gliedern gleich sind - etwa bei der Folge 5, 8, 11, 14, 17, 20. Der Abstand der Glieder beträgt hierbei jeweils 3 und die Folge hat 6 Folgenglieder. Eine arithmetische Folge muss mindestens 3 Folgenglieder haben, kann aber auch unendlich viele haben.

Arithmetische Folgen sind seit Jahrtausenden bekannt und bergen eigentlich keine Geheimnisse mehr. Spannend wird es erst wieder, wenn die Glieder einer arithmetischen Folge noch zusätzliche Eigenschaften haben sollen, wie das bei Primzahlen der Fall ist.

Primzahlen sind ganze Zahlen, die größer als 1 und nur durch 1 und sich selbst ohne Rest teilbar sind. Die zehn kleinsten Primzahlen sind 2, 3, 5, 7, 11, 13, 17, 19, 23 und 29.

Eine arithmetische Primzahlfolge mit fünf Gliedern ist beispielsweise 5, 17, 29, 41, 53. Der Abstand der Zahlen beträgt jeweils 12.

<sup>33</sup>Paul Erdős, ungarischer Mathematiker, 26.03.1913–20.09.1996.

<sup>34</sup>Die Gaussklammer  $\lfloor x \rfloor$  der reellwertigen Zahl  $x$  ist definiert als:  $\lfloor x \rfloor$  ist die größte ganze Zahl kleiner oder gleich  $x$ .

<sup>35</sup>Quellen:

- <http://users.cybercity.dk/~dsl522332/math/aprecords.htm> Original-Quelle
- <http://primes.utm.edu/glossary/page.php?sort=ArithmeticSequence> Original-Quelle
- [http://en.wikipedia.org/wiki/Primes\\_in\\_arithmetic\\_progression](http://en.wikipedia.org/wiki/Primes_in_arithmetic_progression)
- [http://en.wikipedia.org/wiki/Problems\\_involving\\_arithmetic\\_progressions](http://en.wikipedia.org/wiki/Problems_involving_arithmetic_progressions)
- [http://en.wikipedia.org/wiki/Cunningham\\_chain](http://en.wikipedia.org/wiki/Cunningham_chain)
- GEO 10 / 2004: „Experiment mit Folgen“
- <http://www.faz.net> „Hardys Vermutung – Primzahlen ohne Ende“ von Heinrich Hemme (06. Juli 2004)

<sup>36</sup>Arithmetische Folgen mit  $k$  Primzahlen werden auch Prime arithmetic progressions genannt und daher PAP- $k$  bzw. AP- $k$  abgekürzt.

<sup>37</sup>Godfrey Harold Hardy, britischer Mathematiker, 7.2.1877–1.12.1947.

Diese Folge lässt sich nicht verlängern, ohne ihre Eigenschaft einzubüßen, denn das nächste Glied müsste 65 sein, und diese Zahl ist das Produkt aus 5 und 13 und somit keine Primzahl.

Wie viele Glieder kann eine arithmetische Primzahlfolge haben? Mit dieser Frage haben sich schon um 1770 der Franzose Joseph-Louis Lagrange und der Engländer Edward Waring beschäftigt. Im Jahre 1923 vermuteten der berühmte britische Mathematiker Godfrey Harold Hardy und sein Kollege John Littlewood, dass es keine Obergrenze für die Zahl der Glieder gebe. Doch es gelang ihnen nicht, das zu beweisen. Im Jahr 1939 gab es jedoch einen anderen Fortschritt: Der holländische Mathematiker Johannes van der Corput konnte nachweisen, dass es unendlich viele arithmetische Primzahlfolgen mit genau drei Gliedern gibt. Zwei Beispiele hierfür sind 3, 5, 7 und 47, 53, 59.

Die längste Primzahlfolge, die man bisher kennt, hat 25 Glieder. In der Tabelle 3.3 sind die längsten bekannten arithmetischen Primzahlfolgen mit minimaler Distanz<sup>38</sup> aufgelistet.

Den beiden jungen<sup>39</sup> Mathematikern Ben Green und Terence Tao ist es im Jahre 2004 gelungen, die mehr als achtzig Jahre alte Hardysche Vermutung zu beweisen: Es gibt arithmetische Primzahlfolgen beliebiger Länge. Außerdem bewiesen sie, dass es zu jeder vorgegebenen Länge unendlich viele verschiedene solcher Folgen gibt.

Eigentlich hatten Green und Tao nur beweisen wollen, dass es unendlich viele arithmetische Primzahlfolgen mit vier Gliedern gibt. Dazu betrachteten sie Mengen, die neben Primzahlen auch Beinaheprimzahlen enthielten. Das sind Zahlen, die nur wenige Teiler haben - beispielsweise die Halbprimzahlen, die Produkte aus genau zwei Primzahlen sind. Dadurch konnten die beiden Mathematiker ihre Arbeit wesentlich erleichtern, denn über Beinaheprimzahlen gab es schon zahlreiche nützliche Theoreme. Schließlich erkannten sie, dass ihr Verfahren viel mächtiger ist, als sie selbst angenommen hatten, und sie bewiesen damit die Hardysche Vermutung.

Der Beweis von Green und Tao umfasst immerhin 49 Seiten. Tatsächlich beliebig lange arithmetische Primzahlfolgen kann man damit aber nicht finden. Der Beweis ist nicht konstruktiv, sondern ein so genannter Existenzbeweis. Das heißt, die beiden Mathematiker haben „nur“ gezeigt, dass beliebig lange Folgen existieren, aber nicht, wie man sie findet.

Das heißt, in der Menge der natürlichen Primzahlen gibt es zum Beispiel eine Folge von einer Milliarde Primzahlen, die alle den gleichen Abstand haben; und davon gibt es unendlich viele. Diese Folgen liegen aber sehr „weit draußen“.

---

<sup>38</sup>Dagegen sind in [http://en.wikipedia.org/wiki/Primes\\_in\\_arithmetic\\_progression](http://en.wikipedia.org/wiki/Primes_in_arithmetic_progression) die „Largest known AP-k“ aufgelistet. Also ist dort das letzte Folgeelement eine möglichst große Primzahl. Hier jedoch sind die Folgen aufgelistet, die die kleinsten bekannten Differenzen haben – für eine gegebene Folgenlänge.

<sup>39</sup>In seinen Memoiren hat Hardy 1940 geschrieben, dass die Mathematik mehr als alle anderen Wissenschaften und Künste ein Spiel für junge Leute sei.

Der damals 27 Jahre alte Ben Green von der University of British Columbia in Vancouver und der 29 Jahre alte Terence Tao von der University of California in Los Angeles scheinen ihm recht zu geben.

Elemente	Startelement	Abstand	Wann Digits	Entdecker
3	3	2	1	G. Lenaire
4	5	6	2	
5	5	6	2	
6	7	30	1909 3	
7	7	150	1909 3	
.....				
21	28.112.131.522.731.197.609	9.699.690 = 19#	2008 20	Jaroslav Wroblewski
22	166.537.312.120.867	96.599.212.710 = 9.959·19#	2006 15	Markus Frind
23	403.185.216.600.637	2.124.513.401.010 = 9.523·23#	2006 15	Markus Frind,
24	515.486.946.529.943	30.526.020.494.970 = 136.831·23#	2008 16	Raanan Chermoni, Jaroslav Wroblewski
25	6.171.054.912.832.631	81.737.658.082.080 = 366.384·23#	2008 16	Raanan Chermoni, Jaroslav Wroblewski

Tabelle 3.3: Arithmetische Primzahlfolgen mit minimaler Distanz (Stand Jan. 2010)



Wer solche Folgen entdecken möchte, sollte folgendes berücksichtigen. Die Länge der Folge bestimmt den Mindestabstand zwischen den einzelnen Primzahlen. Bei einer Folge mit  $k = 6$  Gliedern muss der Abstand 30 oder ein Vielfaches davon betragen. Die Zahl 30 ergibt sich als das Produkt aller Primzahlen, die kleiner als die Folgenlänge, also kleiner als 6, sind:  $6\# = 5\# = 2 \cdot 3 \cdot 5 = 30$ . Es gilt  $10\# = 7\# = 2 \cdot 3 \cdot 5 \cdot 7 = 210$ . Sucht man Folgen mit der Länge 15, so muss der Abstand mindestens  $15\# = 13\# = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 = 30.030$  betragen.

Daraus ergibt sich, dass die Folgenlänge beliebig groß sein kann, aber der Abstand kann nicht jede beliebige Zahl annehmen: es kann keine arithmetische Primzahlfolge mit dem Abstand 100 geben, denn 100 ist nicht durch die Zahl 3 teilbar.

k	k#
2	2
3	6
5	30
7	210
11	2.310
13	30.030
17	510.510
19	9.699.690
23	223.092.870

Tabelle 3.4: Produkte der ersten Primzahlen  $\leq k$  (genannt k Primorial oder  $k\#$ )

#### Weitere Restriktion an die arithmetische Folge:

Sucht man arithmetische Primzahlfolgen, die die *zusätzliche* Bedingung erfüllen, dass alle Primzahlen in der Folge auch *direkt hintereinander* liegen (consecutive primes sequence<sup>40</sup>), wird es noch etwas schwieriger. Auf der Webseite von Chris Caldwell<sup>41</sup> finden Sie weitere Informationen: Die längste bekannte arithmetische Folge, die nur aus direkt hintereinander liegenden Primzahlen besteht, hat die Länge 10 und der Abstand beträgt

$$10\# = 7\# = 2 \cdot 3 \cdot 5 \cdot 7 = 210$$

### 3.8.2 Unbewiesene Aussagen / Vermutungen zu Primzahlen

- Christian Goldbach<sup>42</sup> vermutete: Jede gerade natürliche Zahl größer 2 lässt sich als die Summe zweier Primzahlen darstellen. Mit Computern ist die **Goldbachsche Vermutung** für alle geraden Zahlen bis  $15 \cdot 10^{17}$  (Stand Juli 2009) verifiziert<sup>43</sup>, aber allgemein noch

<sup>40</sup>Sie werden auch consecutive prime arithmetic progressions genannt und daher CPAP-k bzw. CAP-k abgekürzt.

<sup>41</sup><http://primes.utm.edu/glossary/page.php?sort=ArithmeticSequence>

<sup>42</sup>Christian Goldbach, deutscher Mathematiker, 18.03.1690–20.11.1764.

<sup>43</sup>Dass die Goldbachsche Vermutung wahr ist, d.h. für alle geraden natürlichen Zahlen größer als 2 gilt, wird heute allgemein nicht mehr angezweifelt. Der Mathematiker Jörg Richstein vom Institut für Informatik der Universität Gießen hat 1999 die geraden Zahlen bis 400 Billionen ( $4 \cdot 10^{14}$ ) untersucht ([Richstein1999]) und kein Gegen-

nicht bewiesen<sup>44</sup>.

- Bernhard Riemann<sup>45</sup> stellte eine bisher unbewiesene, aber auch nicht widerlegte Formel für die Verteilung von Primzahlen auf, die die Abschätzung weiter verbessern würde.
- Das Benfordsche Gesetz<sup>46,47</sup> gilt nicht für Primzahlen.

Nach dem Benfordschen Gesetz sind die Ziffern in den Zahlen bestimmter empirischer Datensätze (z.B. über Einwohnerzahlen von Städten, Geldbeträge in der Buchhaltung, Naturkonstanten) ungleichmäßig verteilt: Z.B. ist die Ziffer 1 viel häufiger die erste Ziffer einer Zahl als jede andere.

Welche Datensätze diesem Gesetz gehorchen ist noch nicht vollständig geklärt. Timo Eckhardt untersuchte in seiner Diplomarbeit 2008 ausführlich Eigenschaften von Primzahlen. Unter anderem wurden alle Primzahlen bis 7.052.046.499 mit verschiedenen Stellenwert-Basen dargestellt.

Beim Vergleich der Basen 3 bis 10 ergab sich, dass die Abweichung von Benfords Gesetz bei der Basis 3 am geringsten ist. Für die Basis zehn besteht in etwa eine Gleichverteilung der ersten Ziffern. Bei der Untersuchung größerer Basen ergab sich, dass die Verteilung der ersten Ziffern von Basis zu Basis sehr starke Unterschiede aufweist.

---

beispiel gefunden.

Inzwischen wurden noch umfangreichere Überprüfungen bis vorgenommen: Siehe

<http://www.ieeta.pt/~tos/goldbach.html>,

[http://de.wikipedia.org/wiki/Goldbachsche\\_Vermutung](http://de.wikipedia.org/wiki/Goldbachsche_Vermutung),

<http://primes.utm.edu/glossary/page.php/GoldbachConjecture.html>.

Trotzdem ist das kein allgemeiner Beweis.

Dass die Goldbach'sche Vermutung trotz aller Anstrengungen bis heute nicht bewiesen wurde, fördert allerdings einen Verdacht: Seit den bahnbrechenden Arbeiten des österreichischen Mathematikers Kurt Gödel ist bekannt, dass nicht jeder wahre Satz in der Mathematik auch beweisbar ist (siehe <http://www.mathematik.ch/mathematiker/goedel.html>). Möglicherweise hat Goldbach also Recht, und trotzdem wird nie ein Beweis gefunden werden. Das wiederum lässt sich aber vermutlich auch nicht beweisen.

<sup>44</sup>Der englische Verlag *Faber* und die amerikanische Verlagsgesellschaft *Bloomsbury* publizierten 2000 das 1992 erstmals veröffentlichte Buch „Onkel Petros und die Goldbachsche Vermutung“ von Apostolos Doxiadis (deutsch bei Lübbe 2000 und bei BLT als Taschenbuch 2001). Es ist die Geschichte eines Mathematikprofessors, der daran scheitert, ein mehr als 250 Jahre altes Rätsel zu lösen.

Um die Verkaufszahlen zu fördern, schrieben die beiden Verlage einen Preis von 1 Million USD aus, wenn jemand die Vermutung beweist – veröffentlicht in einer angesehenen mathematischen Fachzeitschrift bis Ende 2004.

Erstaunlicherweise durften nur englische und amerikanische Mathematiker daran teilnehmen.

Die Aussage, die der Goldbach-Vermutung bisher am nächsten kommt, wurde 1966 von Chen Jing-Run bewiesen – in einer schwer nachvollziehbaren Art und Weise: Jede gerade Zahl größer 2 ist die Summe einer Primzahl und des Produkts zweier Primzahlen. Z.B.  $20 = 5 + 3 \cdot 5$ .

Die wichtigsten Forschungsergebnisse zur Goldbachschen Vermutung sind zusammengefasst in dem von Wang Yuan herausgegebenen Band: „Goldbach Conjecture“, 1984, World Scientific Series in Pure Maths, Vol. 4.

Gerade diese Vermutung legt nahe, dass wir auch heute noch nicht in aller Tiefe den Zusammenhang zwischen der Addition und der Multiplikation der natürlichen Zahlen verstanden haben.

<sup>45</sup>Bernhard Riemann, deutscher Mathematiker, 17.9.1826–20.7.1866.

<sup>46</sup>[http://de.wikipedia.org/wiki/Benfordsches\\_Gesetz](http://de.wikipedia.org/wiki/Benfordsches_Gesetz),

<http://www.spiegel.de/wissenschaft/mensch/0,1518,632541,00.html>,

[http://arxiv.org/PS\\_cache/arxiv/pdf/0906/0906.2789v1.pdf](http://arxiv.org/PS_cache/arxiv/pdf/0906/0906.2789v1.pdf).

<sup>47</sup>Didaktische Darstellungen zu Anwendungen von Benfords Gesetz finden sich unter:

- Rüdiger Baumann: „Ziffernanalyse zwecks Betrugsaufdeckung — Beispiel für kompetenzorientierten und kontextbezogenen Informatikunterricht“, in LOGIN, Informatische Bildung und Computer in der Schule, Nr. 154/155, 2008, S. 68-72
- Norbert Hungerbühler: „Benfords Gesetz über führende Ziffern“, März 2007, <http://www.educ.ethz.ch/unt/um/mathe/ana/benford>

### 3.8.3 Offene Fragen zu Primzahlzwillingen

Primzahlzwillinge sind Primzahlen, die den Abstand 2 voneinander haben, zum Beispiel 5 und 7, oder 101 und 103, oder  $1.693.965 \cdot 2^{66.443} \pm 1$ .

Das größte bis heute bekannte Primzahlzwillingspaar mit 100.355 Dezimalstellen<sup>48</sup> ist:

$$65.516.468.355 \cdot 2^{333.333} \pm 1$$

Bemerkung: Primzahltrillinge gibt es dagegen nur eines: 3, 5, 7. Bei allen anderen Dreierpacks aufeinanderfolgender ungerader Zahlen ist immer eine durch 3 teilbar und somit keine Primzahl. Offen ist:

- Gibt es unendlich viele oder eine begrenzte Anzahl von Primzahlzwillingen?
- Gibt es eine Formel für die Anzahl der Primzahlzwillinge pro Intervall?

Einen großen Schritt zur Klärung der ersten Frage machten möglicherweise Dan Goldston, János Pintz und Cem Yildirim im Jahre 2003<sup>49</sup>. Die drei Mathematiker beschäftigten sich mit der Verteilung von Primzahlen. Sie konnten beweisen, dass

$$\liminf_{n \rightarrow \infty} \frac{p_{n+1} - p_n}{\log p_n} = 0,$$

wobei  $p_n$  die  $n$ -te Primzahl bezeichnet.

Dies bedeutet, dass der kleinste Häufungspunkt ( $\liminf$ ) der Folge  $\frac{p_{n+1} - p_n}{\log p_n}$  gleich Null ist.

Ein Punkt heißt Häufungspunkt einer Folge, wenn in jeder noch so kleinen Umgebung um diesen Punkt unendlich viele Folgenglieder liegen.

$\log p_n$  ist in etwa der erwartete Abstand zwischen der Primzahl  $p_n$  und der darauf folgenden Primzahl  $p_{n+1}$ .

Der obige Term besagt also, dass es unendlich viele aufeinander folgende Primzahlen gibt, deren Abstand im Verhältnis zum erwarteten durchschnittlichen Abstand beliebig nah an Null bzw. beliebig klein ist.

Außerdem konnte gezeigt werden, dass für unendlich viele Primzahlen gilt<sup>50</sup>:

$$p_{n+1} - p_n < (\log p_n)^{8/9}$$

Diese Aussagen könnten Grundlage sein um zu beweisen, dass es unendlich viele Primzahlzwillinge gibt.

---

<sup>48</sup><http://primes.utm.edu>, Oktober 2009

<sup>49</sup>D. A. Goldston: „Gaps Between Primes“

<http://www.math.sjsu.edu/~goldston/OberwolfachAbstract.pdf>

Siehe auch:

- D. A. Goldstone: „Are There Infinitely Many Twin Primes?“, <http://www.math.sjsu.edu/~goldston/twinprimes.pdf>
- K. Soundararajan: „Small Gaps Between Prime Numbers: The Work Of Goldston-Pintz-Yildirim“, <http://www.ams.org/bull/2007-44-01/S0273-0979-06-01142-6/S0273-0979-06-01142-6.pdf>

<sup>50</sup>c't 2003, Heft 8, Seite 54

### 3.8.4 Weitere offene Fragestellungen

- Der in Kapitel 3.8.1 erwähnte Beweis zu der Funktion  $f : N \rightarrow Z$  mit  $n \mapsto \lfloor a^{3^n} \rfloor$  garantiert nur die Existenz einer solchen Zahl  $a$ . Wie kann diese Zahl  $a$  bestimmt werden, und wird sie einen Wert haben, so dass die Funktion auch von praktischem Interesse ist?
- Gibt es unendlich viele Mersenne-Primzahlen?
- Gibt es unendlich viele Fermatsche Primzahlen?
- Gibt es einen Polynomialzeit-Algorithmus zur Zerlegung einer Zahl in ihre Primfaktoren (vgl. [Klee1997, S. 167])? Diese Frage kann man auf die folgenden drei Fragestellungen aufsplitten:
  - Gibt es einen Polynomialzeit-Algorithmus, der entscheidet, ob eine Zahl prim ist? Diese Frage wurde durch den AKS-Algorithmus beantwortet (vgl. Kapitel 4.11.5, „PRIMES in P“: Testen auf Primalität ist polynomial).
  - Gibt es einen Polynomialzeit-Algorithmus, mit dem man feststellen kann, aus wieviele Primfaktoren eine zusammengesetzte Zahl besteht (ohne diese Primfaktoren zu berechnen)?
  - Gibt es einen Polynomialzeit-Algorithmus, mit dem sich für eine zusammengesetzte Zahl  $n$  ein nicht-trivialer (d.h. von 1 und von  $n$  verschiedener) Teiler von  $n$  berechnen lässt?<sup>51</sup>

Am Ende von Kapitel 4.11.4, Abschnitt RSA-200 können Sie die Größenordnungen ersehen, für die heutige Algorithmen Ergebnisse bei Primzahltests und bei der Faktorisierung liefern.

### 3.8.5 Kurioses und Interessantes zu Primzahlen<sup>52</sup>

Primzahlen sind nicht nur ein sehr aktives und ernstes mathematisches Forschungsgebiet. Mit ihnen beschäftigen sich Menschen auch hobbymäßig und außerhalb der wissenschaftlichen Forschung.

### Mitarbeiterwerbung bei Google im Jahre 2004

Im Sommer 2004 benutzte die Firma Google die Zahl  $e^{53}$ , um Bewerber zu gewinnen<sup>54</sup>.

Auf einer hervorstechenden Reklamewand im kalifornischen Silicon Valley erschien am 12. Juli das folgende geheimnisvolle Rätsel:

---

<sup>51</sup>Vergleiche auch Kapitel 4.11.5 und Kapitel 4.11.4.

<sup>52</sup>Weitere kuriose und seltsame Dinge zu Primzahlen finden sich auch unter:

- <http://primes.utm.edu/curios/home.php>
- <http://www.primzahlen.de/files/theorie/index.htm>

<sup>53</sup>Die Basis des natürlichen Logarithmus  $e$  ist ungefähr 2,718 281 828 459. Dies ist eine der bedeutendsten Zahlen in der Mathematik. Sie wird gebraucht für komplexe Analysis, Finanzmathematik, Physik und Geometrie. Nun wurde sie – meines Wissens – das erste Mal für Marketing oder Personalbeschaffung verwendet.

<sup>54</sup>Die meisten Informationen für diesen Paragraphen stammen aus dem Artikel „e-number crunching“ von John Allen Paulos in TheGuardian vom 30.09.2004 und aus dem Internet:

- <http://www.mkaz.com/math/google/>
- <http://epramono.blogspot.com/2004/10/7427466391.html>
- <http://mathworld.wolfram.com/news/2004-10-13/google/>
- <http://www.math.temple.edu/~paulos/>

*(first 10-digit prime found in consecutive digits of e).com*

In der Dezimaldarstellung von  $e$  sollte man also die erste 10-stellige Primzahl finden, die sich in den Nachkommastellen befindet. Mit verschiedenen Software-Tools kann man die Antwort finden:

7.427.466.391

Wenn man dann die Webseite *www.7427466391.com* besuchte, bekam man ein noch schwierigeres Rätsel gezeigt. Wenn man auch dieses zweite Rätsel löste, kam man zu einer weiteren Webseite, die darum bat, den Lebenslauf an Google zu senden. Diese Werbekampagne erzielte eine hohe Aufmerksamkeit.

Wahrscheinlich nahm Google an, dass man gut genug ist, für sie zu arbeiten, wenn man diese Rätsel lösen kann. Aber bald konnte jeder mit Hilfe der Google-Suche ohne Anstrengung die Antworten finden, weil viele die Lösung der Rätsel ins Netz gestellt hatten.<sup>55</sup>

## **Contact [Regie Robert Zemeckis, 1997] – Primzahlen zur Kontaktaufnahme**

Der Film entstand nach dem gleichnamigen Buch von Carl Sagan.

Die Astronomin Dr. Ellie Arroway (Jodie Foster) entdeckt nach jahrelanger vergeblicher Suche Signale vom 26 Lichtjahre entfernten Sonnensystem Wega. In diesen Signalen haben Außerirdische die Primzahlen lückenlos in der richtigen Reihenfolge verschlüsselt. Daran erkennt die Heldin, dass diese Nachricht anders ist als die ohnehin ständig auf der Erde eintreffenden Radiowellen, die kosmischer und zufälliger Natur sind (von Radiogalaxien, Pulsaren oder Quasaren). In einer entlarvenden Szene fragt sie daraufhin ein Politiker, warum diese intelligenten Wesen nicht gleich Englisch sprechen ...

Eine Kommunikation mit absolut fremden und unbekannten Wesen aus dem All ist aus 2 Gründen sehr schwierig: Zunächst kann man sich bei der hier angenommenen Entfernung und dem damit verbundenen langen Übertragungsweg in einem durchschnittlichen Menschenleben nur einmal in jeder Richtungen nacheinander austauschen. Zum Zweiten muss man für den Erstkontakt darauf achten, dass eine möglichst hohe Chance besteht, dass der Empfänger der Radiowellen die Botschaft überhaupt bemerkt und dass er sie als Nachricht von intelligenten Wesen einstuft. Deshalb senden die Außerirdischen am Anfang ihrer Nachricht Zahlen, die als der einfachste Teil jeder höheren Sprache angesehen werden können und die nicht ganz trivial sind: die Folge der Primzahlen. Diese speziellen Zahlen spielen in der Mathematik eine so fundamentale Rolle, dass man annehmen kann, dass sie jeder Spezies vertraut sind, die das technische Know-how hat, Radiowellen zu empfangen.

Die Aliens schicken danach den Plan zum Bau einer mysteriösen Maschine ...

---

<sup>55</sup>Im zweiten Level der Rätsel musste man das 5. Glied einer gegebenen Zahlenfolge finden: Dies hatte nichts mehr mit Primzahlen zu tun.

### 3.9 Anhang: Anzahl von Primzahlen in verschiedenen Intervallen

Zehnerintervall		Hunderterintervall		Tausenderintervall	
Intervall	Anzahl	Intervall	Anzahl	Intervall	Anzahl
1-10	4	1-100	25	1-1.000	168
11-20	4	101-200	21	1.001-2.000	135
21-30	2	201-300	16	2.001-3.000	127
31-40	2	301-400	16	3.001-4.000	120
41-50	3	401-500	17	4.001-5.000	119
51-60	2	501-600	14	5.001-6.000	114
61-70	2	601-700	16	6.001-7.000	117
71-80	3	701-800	14	7.001-8.000	107
81-90	2	801-900	15	8.001-9.000	110
91-100	1	901-1.000	14	9.001-10.000	112

Tabelle 3.5: Wieviele Primzahlen gibt es innerhalb der ersten Zehnerintervalle?

Intervall	Anzahl	Durchschnittl. Anzahl pro 1000
1 - 10.000	1.229	122,900
1 - 100.000	9.592	95,920
1 - 1.000.000	78.498	78,498
1 - 10.000.000	664.579	66,458
1 - 100.000.000	5.761.455	57,615
1 - 1.000.000.000	50.847.534	50,848
1 - 10.000.000.000	455.052.512	45,505

Tabelle 3.6: Wieviele Primzahlen gibt es innerhalb der ersten Dimensionsintervalle?

Eine Visualisierung der Anzahl von Primzahlen in höheren Intervallen von Zehnerpotenzen finden sich in Kapitel 3.13 auf Seite 84.

### 3.10 Anhang: Indizierung von Primzahlen ( $n$ -te Primzahl)

Index	Genauer Wert	Gerundeter Wert	Bemerkung
1	2	2	
2	3	3	
3	5	5	
4	7	7	
5	11	11	
6	13	13	
7	17	17	
8	19	19	
9	23	23	
10	29	29	
100	541	541	Alle Primzahlen bis zu 1E+07 waren am Beginn des 20. Jahrhunderts bekannt.
1000	7917	7917	
664.559	9.999.991	9,99999E+06	
1E+06	15.485.863	1,54859E+07	
6E+06	104.395.301	1,04395E+08	
1E+07	179.424.673	1,79425E+08	
1E+09	22.801.763.489	2,28018E+10	
1E+12	29.996.224.275.833	2,99962E+13	
			Diese Primzahl wurde 1959 entdeckt.

Tabelle 3.7: Liste selektierter  $n$ -ter Primzahlen

**Bemerkung:** Mit Lücke wurden früh sehr große Primzahlen entdeckt.

#### Web-Links (URLs):

<http://www.math.Princeton.EDU/~arbooker/nthprime.html>

[http://www.utm.edu/research/primes/notes/by\\_year.html](http://www.utm.edu/research/primes/notes/by_year.html).

### 3.11 Anhang: Größenordnungen / Dimensionen in der Realität

Bei der Beschreibung kryptographischer Protokolle und Algorithmen treten Zahlen auf, die so groß bzw. so klein sind, dass sie einem intuitiven Verständnis nicht zugänglich sind. Es kann daher nützlich sein, Vergleichszahlen aus der uns umgebenden realen Welt bereitzustellen, so dass man ein Gefühl für die Sicherheit kryptographischer Algorithmen entwickeln kann. Die angegebenen Werte stammen größtenteils aus [Schwenk1996] und [Schneier1996, S.18].

Wahrscheinlichkeit, dass Sie auf ihrem nächsten Flug entführt werden	$5,5 \cdot 10^{-6}$
Jährliche Wahrscheinlichkeit, von einem Blitz getroffen zu werden	$10^{-7}$
Wahrscheinlichkeit für 6 Richtige im Lotto	$7,1 \cdot 10^{-8}$
Risiko, von einem Meteoriten erschlagen zu werden	$1,6 \cdot 10^{-12}$
Zeit bis zur nächsten Eiszeit (in Jahren)	$14.000 = (2^{14})$
Zeit bis die Sonne verglüht (in Jahren)	$10^9 = (2^{30})$
Alter der Erde (in Jahren)	$10^9 = (2^{30})$
Alter des Universums (in Jahren)	$10^{10} = (2^{34})$
Anzahl der Moleküle in einem Wassertropfen	$10^{20} = (2^{63})$
Anzahl der auf der Erde lebenden Bakterien	$10^{30,7} = (2^{102})$
Anzahl der Atome der Erde	$10^{51} = (2^{170})$
Anzahl der Atome der Sonne	$10^{57} = (2^{190})$
Anzahl der Atome im Universum (ohne dunkle Materie)	$10^{77} = (2^{265})$
Volumen des Universums (in $cm^3$ )	$10^{84} = (2^{280})$

Tabelle 3.8: Wahrscheinlichkeiten und Größenordnungen aus Physik und Alltag



### 3.12 Anhang: Spezielle Werte des Zweier- und Zehnersystems

Diese Werte können dazu benutzt werden, aus einer Schlüssellänge in Bit die Anzahl möglicher Schlüssel und den Such-Aufwand abzuschätzen (wenn man z.B. annimmt, dass 1 Million Schlüssel in 1 sec durchprobiert werden können).

Dualsystem	Zehnersystem
$2^{10}$	1024
$2^{40}$	$1,09951 \cdot 10^{12}$
$2^{56}$	$7,20576 \cdot 10^{16}$
$2^{64}$	$1,84467 \cdot 10^{19}$
$2^{80}$	$1,20893 \cdot 10^{24}$
$2^{90}$	$1,23794 \cdot 10^{27}$
$2^{112}$	$5,19230 \cdot 10^{33}$
$2^{128}$	$3,40282 \cdot 10^{38}$
$2^{150}$	$1,42725 \cdot 10^{45}$
$2^{160}$	$1,46150 \cdot 10^{48}$
$2^{192}$	$6,27710 \cdot 10^{57}$
$2^{250}$	$1,80925 \cdot 10^{75}$
$2^{256}$	$1,15792 \cdot 10^{77}$
$2^{320}$	$2,13599 \cdot 10^{96}$
$2^{512}$	$1,34078 \cdot 10^{154}$
$2^{768}$	$1,55252 \cdot 10^{231}$
$2^{1024}$	$1,79769 \cdot 10^{308}$
$2^{2048}$	$3,23170 \cdot 10^{616}$

Tabelle 3.9: Spezielle Werte des Zweier- und Zehnersystems

Solche Tabellen lassen sich mit Computeralgebrasystemen einfach berechnen. Hier ein Codebeispiel für das CAS-System Sage:

---

#### Sage-Beispiel 3.1 Spezielle Werte des Zweier- und Zehnersystems

---

```
E = [10, 40, 56, 64, 80, 90, 112, 128, 150, 160, 192, 256, 1024, 2048]
for e in E:
    # print "2^" + str(e), "---", 1.0*(2^e)
    print "2^%4d" % e , " --- ", RR(2^e).n(24)
....:
2^ 10 --- 1024.00
2^ 40 --- 1.09951e12
2^ 56 --- 7.20576e16
2^ 64 --- 1.84467e19
2^ 80 --- 1.20893e24
2^ 90 --- 1.23794e27
2^ 112 --- 5.19230e33
2^ 128 --- 3.40282e38
2^ 150 --- 1.42725e45
2^ 160 --- 1.46150e48
2^ 192 --- 6.27710e57
2^ 256 --- 1.15792e77
2^1024 --- 1.79769e308
2^2048 --- 3.23170e616
```

---

### 3.13 Anhang: Visualisierung der Menge der Primzahlen in hohen Bereichen

#### Zur Verteilung von Primzahlen

Zwischen 1 und 10 gibt es 4 Primzahlen. Zwischen  $10^3$  und  $10^4$  sind es schon 1.061. Im Intervall  $[10^9, 10^{10}]$  liegen  $404.204.977 \approx 4 \cdot 10^8$  Primzahlen und in dem Intervall von  $10^{19}$  bis  $10^{20}$  sind es schon  $1.986.761.935.284.574.233 \approx 1,9 \cdot 10^{18}$  Primzahlen<sup>56</sup>.

Warum unterscheidet sich die Anzahl der Primzahlen, die in den verschiedenen Intervallen liegen so stark, obwohl sich die Intervallgrenzen jeweils nur um den Wert 1 im Exponenten der 10-er Potenz unterscheiden?

#### Primzahlsatz

Die Anzahl der Primzahlen bis zu einer gegebenen Zahl  $x$  kann durch eine Formel, den sogenannten Primzahlsatz, näherungsweise bestimmt werden.  $\pi(x)$  bezeichne die Anzahl der Primzahlen kleiner oder gleich der Zahl  $x$ . Dann lautet die Formel

$$\pi(x) \sim \frac{x}{\ln x}.$$

Es ist zu beachten, dass diese Formel die Anzahl der Primzahlen kleiner oder gleich  $x$  nur ungefähr angibt. Sie wird jedoch genauer je größer  $x$  wird. Im folgenden wollen wir den Primzahlsatz nutzen, um uns die Verteilung der Primzahlen anzusehen.

Um zu verstehen, warum die Anzahl der Primzahlen so schnell wächst, obwohl sich die Intervallgrenzen jeweils nur um einen Exponentenwert von 1 unterscheiden, werfen wir einen Blick auf beide Komponenten der rechten Seite der Formel:  $x$  und  $\ln x$ .

#### Die Funktionen $x$ und $10^x$

Die Funktion  $x$  ist eine Gerade. Sie ist in Abbildung 3.1a auf Seite 85 zu sehen.

Als nächstes tragen wir die Funktion der Intervallgrenzen in einem Graphen ab, der in Abbildung 3.1b auf Seite 85 zu finden ist. Um einen Eindruck zu bekommen, wie die Funktionen aussehen, wurde der Definitionsbereich von 0 bis  $10^{10}$  bzw. entsprechend von 0 bis 10 gewählt. Es ist zu sehen, dass mit steigendem Exponenten  $x$  die Zahlen immer größer werden.

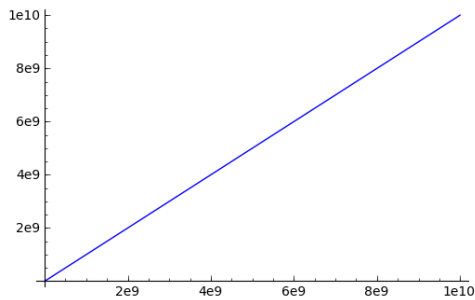
#### Die Funktion $\ln x$

Im Vergleich dazu betrachten wir die Funktion  $\ln x$ . Auf dem linken Bild von Abbildung 3.2 auf Seite 85 wurde der Definitionsbereich von 1 bis 100 gewählt. Das rechte Bild zeigt die Werte der Funktion bis  $10^{10}$ .

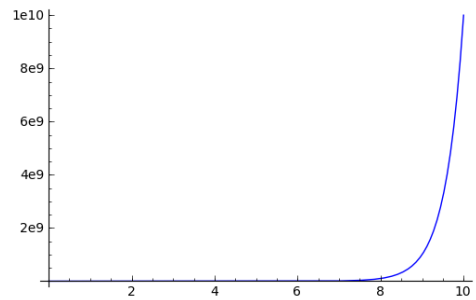
Es ist gut zu erkennen, dass die Werte der Funktion  $\ln x$  im Vergleich zu  $x$  langsam wachsen. Dies verdeutlicht auch der Graph beider Funktionen in Abbildung 3.3 auf Seite 85. Zudem wurde in diesen Graphen auch die Funktion  $\frac{x}{\ln x}$  eingezeichnet.

---

<sup>56</sup><http://de.wikipedia.org/wiki/Primzahlsatz>

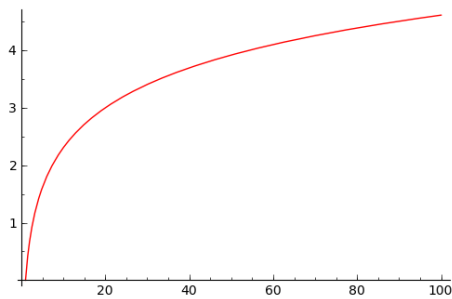


(a)  $x$

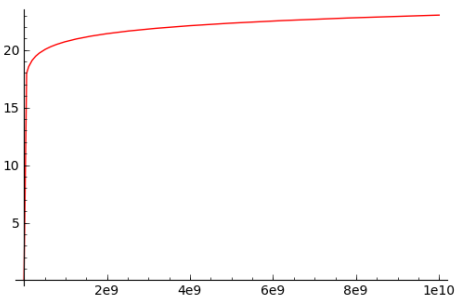


(b)  $10^x$

Abbildung 3.1: Graph der Funktionen  $x$  und  $10^x$



(a)



(b)

Abbildung 3.2: Graph der Funktion  $\ln x$  bis 100 und bis  $10^{10}$

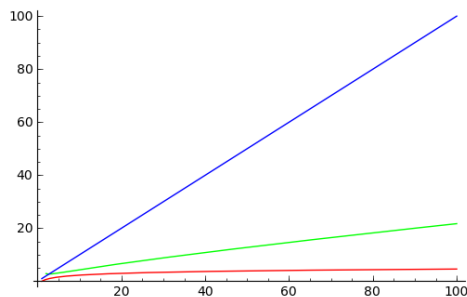


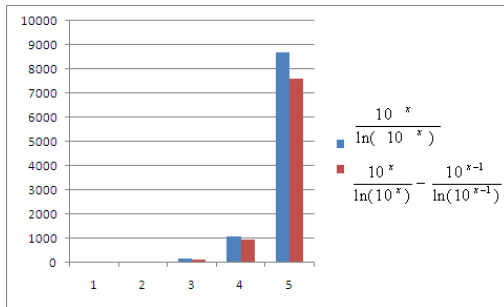
Abbildung 3.3: Die Funktionen  $x$  (blau),  $\ln x$  (rot) und  $\frac{x}{\ln x}$  (grün)

### Die Funktion $\frac{x}{\ln x}$

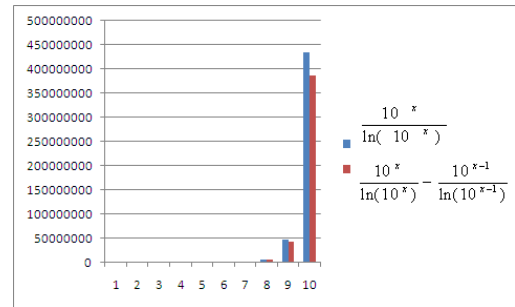
Die Funktion  $\frac{x}{\ln x}$  setzt sich also zusammen aus der Funktion  $x$  im Zähler und der im Verhältnis dazu sehr langsam wachsenden Funktion  $\ln x$  im Nenner. Die Anzahl der Primzahlen kleiner oder gleich einer Zahl  $x$  ist zwar verhältnismäßig klein im Vergleich zu  $x$  selbst. Dennoch ist  $\frac{x}{\ln x}$  eine wachsende Funktion. Dies wird auch deutlich in Abbildung 3.3 auf Seite 85.

## Die Anzahl der Primzahlen in verschiedenen Intervallen

Um zu sehen, wie sich die Anzahl der Primzahl in den Intervallen  $[10^{x-1}, 10^x]$  verhalten, werfen wir einen Blick auf Abbildung 3.4 auf Seite 86. Hier ist  $\frac{10^x}{\ln 10^x}$  im Vergleich zu  $\frac{10^x}{\ln 10^x} - \frac{10^{x-1}}{\ln 10^{x-1}}$  abgebildet. Zum einen für Werte des Exponenten  $x$  von 1 bis 5 zum anderen für  $x$  von 1 bis 10. Die blauen Balken geben an, wieviele Primzahlen es insgesamt bis  $10^x$  gibt. Die roten Balken zeigen, wieviele Primzahlen jeweils im Intervall  $[10^{x-1}, 10^x]$  liegen. Dies verdeutlicht, dass die Anzahl der Primzahlen in Intervallen mit höheren Exponenten immer noch stark steigt.



(a)



(b)

Abbildung 3.4: Anzahl der Primzahlen im Intervall 0 bis  $10^x$  (blau) und im Interall  $[10^{x-1}, 10^x]$  (rot) (für verschiedene Exponenten  $x$ ).

Eine Tabelle mit den Anzahlen von Primzahlen in einigen ausgewählten Intervallen finden Sie in Kapitel 3.9 auf Seite 80.

---

### Sage-Beispiel 3.2 Erzeugen der Graphen zu den drei Funktionen $x$ , $\log(x)$ und $x/\log(x)$

---

```
# Definition der Funktion f(x)=x und Plots für die Definitionsbereiche 0 bis 10^10 und 0 bis 100
sage: def f(x):return x
.....:
sage: F=plot(f,(0,10^10))
sage: F.plot()

sage: F2=plot(f,(1,100))
sage: F2.plot()

# Definition der Funktion g(x)=10^x und Plot für den Definitionsbereich 0 bis 10
sage: def g(x): return 10^x
.....:
sage: G=plot(g,(0,10))
sage: G.plot()

# Definition der Funktion h(x)=log(x) und Plots für die Definitionsbereiche 1 bis 100 und 1 bis 10^10
sage: def h(x): return log(x)
.....:
sage: H=plot(h,(1,100),color="red")
sage: H.plot()

sage: H2=plot(h,(1,10^10),color="red")
sage: H2.plot()

# Definition der Funktion k(x)=x/log(x) und Plot für den Definitionsbereich 2 bis 100
sage: def k(x): return x/log(x)
.....:
sage: K=plot(k,(2,100),color="green")
sage: K.plot()

# Plot der Funktionen f, k und h für den Definitionsbereich von 1 bzw. 2 bis 100
sage: F2+K+H

# Generieren der Daten für die Balkencharts .....
# Bestimmen der Anzahl der Primzahlen im Intervall [1,10]
sage: pari(10).primepi()-pari(1).primepi()
4

# Bestimmen der Anzahl der Primzahlen im Intervall [10^3,10^4]
sage: pari(10**4).primepi()-pari(10**3).primepi()
1061

# Bestimmen der Anzahl der Primzahlen im Intervall [10^8,10^9]
sage: pari(10**9).primepi()-pari(10**8).primepi()
45086079

# (ab 10^10: OverflowError: long int too large to convert)
```

---

## 3.14 Anhang: Beispiele mit Sage

Der folgende Abschnitt enthält Sage Source-Code zu den Inhalten aus Kapitel 3 („Primzahlen“).

### 3.14.1 Einfache Funktionen zu Primzahlen mit Sage

In diesem Teil des Anhangs finden Sie den Quellcode für Sage, mit dem man einige einfache Berechnungen zu Primzahlen durchführen kann<sup>57</sup>.

---

**Sage-Beispiel 3.3** Einige einfache Funktionen zu Primzahlen

---

```
# Primzahlen (allgemeine Befehle)
# Die Menge der Primzahlen
sage: P=Primes(); P
Set of all prime numbers: 2, 3, 5, 7, ...

# Gibt die nächste Primzahl aus
sage: next_prime(5)
7

# Gibt die Anzahl der Primzahlen <=x aus
sage: pari(10).primepi()
4

# Gibt die ersten x Primzahlen aus
sage: primes_first_n(5)
[2, 3, 5, 7, 11]

# Gibt die Primzahlen in einem Interval aus
sage: list(primes(1,10))
[2, 3, 5, 7]
```

---

---

<sup>57</sup> Siehe die Sage-Dokumentation über Elementare Zahlentheorie [http://www.sagemath.org/doc/constructions/number\\_theory.html](http://www.sagemath.org/doc/constructions/number_theory.html).

### 3.14.2 Primalitäts-Check der von einer quadratischen Funktion erzeugten Zahlen

In diesem Anhang finden Sie den Quellcode für Sage, mit dem man z.B. prüfen kann, ob die von der Funktion  $f(n) = n^2 - 9n + 61$  erzeugten Zahlen prim sind. Der Quellcode definiert eine Funktion namens `quadratic_prime_formula()`, die die folgenden drei Argumente hat:

- **start** — Ein natürliche Zahl, die die Untergrenze der Folge `start, start + 1, start + 2, ..., end - 1, end` darstellt.
- **end** — Ein natürliche Zahl, die die Obergrenze der Folge `start, start + 1, start + 2, ..., end - 1, end` darstellt.
- **verbose** — (Standardwert: `True`) Ein Schalter, der festlegt, ob für jede erzeugte Einzelzahl  $f(n)$  eine Ausgabe zur Primalität erfolgen soll.

Eine sinnvolle Änderung dieses Codes besteht darin, eine andere Funktion anzugeben, deren Funktionswerte auf Primalität geprüft werden sollen.

---

#### Sage-Beispiel 3.4 Testen der Primalität von Funktionswerten, erzeugt von einer quadratischen Funktion

---

```
def quadratic_prime_formula(start, end, verbose=True):
    print "N -- N^2 - 9*N + 61"
    P = 0 # the number of primes between start and end
    for n in xrange(start, end + 1):
        X = n^2 - 9*n + 61
        if is_prime(X):
            P += 1
            if verbose:
                print str(n) + " -- " + str(X) + " is prime"
        else:
            if verbose:
                print str(n) + " -- " + str(X) + " is NOT prime"
    print "Number of primes: " + str(P)
    print "Percentage of primes: " + str(float((P * 100) / (end - start + 1)))
```

---

Mit dem folgenden Funktionsaufruf berechnen wir die Werte von  $f(n) = n^2 - 9n + 61$  für  $n = 0, 1, 2, \dots, 50$  und verifizieren die Primalität der erzeugten Funktionswerte:

```
sage: quadratic_prime_formula(0, 50)
N -- N^2 - 9*N + 61
0 -- 61 is prime
1 -- 53 is prime
2 -- 47 is prime
3 -- 43 is prime
4 -- 41 is prime
5 -- 41 is prime
6 -- 43 is prime
7 -- 47 is prime
8 -- 53 is prime
9 -- 61 is prime
10 -- 71 is prime
11 -- 83 is prime
12 -- 97 is prime
13 -- 113 is prime
```

```

14 -- 131 is prime
15 -- 151 is prime
16 -- 173 is prime
17 -- 197 is prime
18 -- 223 is prime
19 -- 251 is prime
20 -- 281 is prime
21 -- 313 is prime
22 -- 347 is prime
23 -- 383 is prime
24 -- 421 is prime
25 -- 461 is prime
26 -- 503 is prime
27 -- 547 is prime
28 -- 593 is prime
29 -- 641 is prime
30 -- 691 is prime
31 -- 743 is prime
32 -- 797 is prime
33 -- 853 is prime
34 -- 911 is prime
35 -- 971 is prime
36 -- 1033 is prime
37 -- 1097 is prime
38 -- 1163 is prime
39 -- 1231 is prime
40 -- 1301 is prime
41 -- 1373 is prime
42 -- 1447 is prime
43 -- 1523 is prime
44 -- 1601 is prime
45 -- 1681 is NOT prime
46 -- 1763 is NOT prime
47 -- 1847 is prime
48 -- 1933 is prime
49 -- 2021 is NOT prime
50 -- 2111 is prime
Number of primes: 48
Percentage of primes: 94.1176470588

```

Die Statistik in den letzten beiden Zeilen der Ausgabe zeigt, dass  $f(n)$  48 Primzahlen erzeugt für  $0 \leq n \leq 50$ , was ca. 94% der von  $f(n)$  erzeugten Werte darstellt.

Bei längeren Sequenzen ist die Einzelausgabe der erzeugten Zahl und ihrer Primalität ziemlich unpraktisch. Deshalb wird in der folgenden Sage-Session nur die Statistik am Ende ausgegeben: die Gesamtzahl und der prozentuale Anteil der Primzahlen, die von  $f(n)$  im Bereich  $0 \leq n \leq 1000$  erzeugt wurde.

```

sage: quadratic_prime_formula(0, 1000, verbose=False)
N -- N^2 - 9*N + 61
Number of primes: 584
Percentage of primes: 58.3416583417

```



# Literaturverzeichnis

- [Aaronson2003] Scott Aaronson,  
*The Prime Facts: From Euclid to AKS*,  
<http://www.scottaaronson.com/writings/prime.pdf>  
Dieses schöne Paper wurde mir erst nach Vollendung dieses Artikels bekannt: Es bietet ebenfalls einen einfachen, didaktisch gut aufbereiteten und trotzdem nicht oberflächlichen Einstieg in dieses Thema (ist nur in Englisch verfügbar).
- [Bartholome1996] A. Bartholomé, J. Rung, H. Kern,  
*Zahlentheorie für Einsteiger*, Vieweg 1995, 2. Auflage 1996.
- [Blum1999] W. Blum,  
*Die Grammatik der Logik*, dtv, 1999.
- [Bundschuh1998] Peter Bundschuh,  
*Einführung in die Zahlentheorie*, Springer 1988, 4. Auflage 1998.
- [Crandell2001] Richard Crandell, Carl Pomerance,  
*Prime Numbers. A Computational Perspective*, Springer, 2001.
- [Doxiadis2000] Apostolos Doxiadis,  
*Onkel Petros und die Goldbachsche Vermutung*,  
Bloomsbury 2000 (deutsch bei Lübbe 2000 und bei BLT als Taschenbuch 2001).
- [Graham1989] R.E. Graham, D.E. Knuth, O. Patashnik,  
*Concrete Mathematics*, Addison-Wesley, 1989.
- [Klee1997] V. Klee, S. Wagon,  
*Ungelöste Probleme in der Zahlentheorie und der Geometrie der Ebene*,  
Birkhäuser Verlag, 1997.
- [Knuth1981] Donald E. Knuth,  
*The Art of Computer Programming, vol 2: Seminumerical Algorithms*,  
Addison-Wesley, 1969, 2nd edition 1981.
- [Lorenz1993] F. Lorenz,  
*Algebraische Zahlentheorie*, BI Wissenschaftsverlag, 1993.
- [Oppliger2005] Rolf Oppliger  
*Contemporary Cryptography*, Artech House, 2005,  
<http://www.esecurity.ch/Books/cryptography.html>.
- [Padberg1996] F. Padberg,  
*Elementare Zahlentheorie*, Spektrum Akademischer Verlag 1988, 2. Auflage 1996.

- [Pieper1983] H. Pieper,  
*Zahlen aus Primzahlen*, Verlag Harri Deutsch 1974, 3. Auflage 1983.
- [Richstein1999] J. Richstein,  
*Verifying the Goldbach Conjecture up to  $4 \cdot 10^{14}$* , Mathematics of Computation 70, 2001,  
 S. 1745-1749.
- [Scheid1994] Harald Scheid,  
*Zahlentheorie*, BI Wissenschaftsverlag, 2. Auflage, 1994.
- [Schneier1996] Bruce Schneier,  
*Applied Cryptography, Protocols, Algorithms, and Source Code in C*,  
 Wiley and Sons, 2nd edition 1996.
- [Schroeder1999] M.R. Schroeder,  
*Number Theory in Science and Communication*,  
 Springer 1984, 3rd edition 1997, Corrected Printing 1999.
- [Schwenk1996] J. Schwenk  
*Conditional Access*, in taschenbuch der telekom praxis 1996,  
 Hrgb. B. Seiler, Verlag Schiele und Schön, Berlin.
- [Shoup2005] Victor Shoup  
*A Computational Introduction to Number Theory and Algebra*, Cambridge University  
 Press, 2005,  
<http://shoup.net/ntb/>.
- [Tietze1973] H. Tietze,  
*Gelöste und ungelöste mathematische Probleme*, Verlag C.H. Beck 1959, 6. Auflage 1973.

# Web-Links

1. GIMPS (Great Internet Mersenne-Prime Search)  
www.mersenne.org ist die Homepage des GIMPS-Projekts,  
<http://www.mersenne.org/prime.htm>
2. Die Proth Search Page mit dem Windows-Programm von Yves Gallot  
<http://www.utm.edu/research/primes/programs/gallot/index.html>
3. Verallgemeinerte Fermat-Primzahlen-Suche  
<http://primes.utm.edu/top20/page.php?id=12>
4. Verteilte Suche nach Teilern von Fermatzahlen  
<http://www.fermatsearch.org/>
5. An der Universität von Tennessee findet man umfangreiche Forschungsergebnisse über Primzahlen.  
<http://www.utm.edu/>
6. Den besten Überblick zu Primzahlen (weil sehr aktuell und vollständig) bieten m.E. „The Prime Pages“ von Professor Chris Caldwell.  
<http://www.utm.edu/research/primes>
7. Beschreibungen u.a. zu Primzahltests  
<http://www.utm.edu/research/primes/mersenne.shtml>  
<http://www.utm.edu/research/primes/prove/index.html>
8. Ausgabe der  $n$ -ten Primzahl  
[http://www.utm.edu/research/primes/notes/by\\_year.html](http://www.utm.edu/research/primes/notes/by_year.html)
9. Der Supercomputerhersteller SGI Cray Research beschäftigte nicht nur hervorragende Mathematiker, sondern benutzte die Primzahltests auch als Benchmarks für seine Maschinen.  
[http://www.isthe.com/chongo/tech/math/prime/prime\\_press.html](http://www.isthe.com/chongo/tech/math/prime/prime_press.html)
10. Das Cunningham-Projekt,  
<http://www.cerias.purdue.edu/homes/ssw/cun/>
11. <http://www.eff.org/awards/coop>
12. <http://www.math.Princeton.EDU/~arbooker/nthprime.html>
13. Goldbach conjecture verification project von Tomás Oliveira e Silva,  
<http://www.ieeta.pt/~tos/goldbach.html>
14. <http://www.mathematik.ch/mathematiker/goedel.html>

## **Dank**

Für das sehr konstruktive Korrekturlesen der ersten Versionen dieses Artikels: Hr. Henrik Koy und Hr. Roger Oyono.

## Kapitel 4

# Einführung in die elementare Zahlentheorie mit Beispielen

(Bernhard Esslinger, Juli 2001; Updates: Nov. 2001, Juni 2002, Mai 2003, Mai 2005, März 2006, Juni 2007, Juli 2009, Jan. 2010)

Diese „Einführung“ bietet einen Einstieg für mathematisch Interessierte. Erforderlich sind nicht mehr Vorkenntnisse als die, die im Grundkurs Mathematik am Gymnasium vermittelt werden.

Wir haben uns bewusst an „Einsteigern“ und „Interessenten“ orientiert, und nicht an den Gepflogenheiten mathematischer Lehrbücher, die auch dann „Einführung“ genannt werden, wenn sie schon auf der 5. Seite nicht mehr auf Anhieb zu verstehen sind und sie eigentlich den Zweck haben, dass man danach auch spezielle Monographien zu dem Thema lesen können soll.

### 4.1 Mathematik und Kryptographie

Ein großer Teil der modernen, asymmetrischen Kryptographie beruht auf mathematischen Erkenntnissen – auf den Eigenschaften („Gesetzen“) ganzer Zahlen, die in der elementaren Zahlentheorie untersucht werden. „Elementar“ bedeutet hier, dass die zahlentheoretischen Fragestellungen im wesentlichen in der Menge der natürlichen und der ganzen Zahlen durchgeführt werden.

Weitere mathematische Disziplinen, die heute in der Kryptographie Verwendung finden, sind (vgl. [Bauer1995, S. 2], [Bauer2000, Seite 3]) :

- Gruppentheorie
- Kombinatorik
- Komplexitätstheorie
- Ergodentheorie
- Informationstheorie.

Die Zahlentheorie oder Arithmetik (hier wird mehr der Aspekt des Rechnens mit Zahlen betont) wurde von Carl Friedrich Gauss<sup>1</sup> als besondere mathematische Disziplin begründet.

---

<sup>1</sup>Carl Friedrich Gauss, deutscher Mathematiker und Astronom, 30.4.1777–23.2.1855.

Zu ihren elementaren Gegenständen gehören: größter gemeinsamer Teiler<sup>2</sup> (ggT), Kongruenzen (Restklassen), Faktorisierung, Satz von Euler-Fermat und primitive Wurzeln. Kernbegriff sind jedoch die Primzahlen und ihre multiplikative Verknüpfung.

Lange Zeit galt gerade die Zahlentheorie als Forschung pur, als Paradebeispiel für die Forschung im Elfenbeinturm. Sie erforschte die „geheimnisvollen Gesetze im Reich der Zahlen“ und gab Anlass zu philosophischen Erörterungen, ob sie beschreibt, was überall in der Natur schon da ist, oder ob sie ihre Elemente (Zahlen, Operatoren, Eigenschaften) nicht künstlich konstruiert.

Inzwischen weiß man, dass sich zahlentheoretische Muster überall in der Natur finden. Zum Beispiel verhalten sich die Anzahl der links- und der rechtsdrehenden Spiralen einer Sonnenblume zueinander wie zwei aufeinanderfolgende Fibonacci-Zahlen<sup>3</sup>, also z.B. wie 21 : 34.

Außerdem wurde spätestens mit den zahlentheoretischen Anwendungen der modernen Kryptographie klar, dass eine jahrhundertlang als theoretisch geltende Disziplin praktische Anwendung findet, nach deren Experten heute eine hohe Nachfrage auf dem Arbeitsmarkt besteht.

Anwendungen der (Computer-)Sicherheit bedienen sich heute der Kryptographie, weil Kryptographie als mathematische Disziplin einfach besser und beweisbarer ist als alle im Laufe der Jahrhunderte erfundenen „kreativen“ Verfahren der Substitution und besser als alle ausgefeilten physischen Techniken wie beispielsweise beim Banknotendruck [Beutelspacher1996, S. 4].

In diesem Artikel werden in einer leicht verständlichen Art die grundlegenden Erkenntnisse der elementaren Zahlentheorie anhand vieler Beispiele vorgestellt – auf Beweise wird (fast) vollständig verzichtet (diese finden sich in den mathematischen Lehrbüchern).

Ziel ist nicht die umfassende Darstellung der zahlentheoretischen Erkenntnisse, sondern das Aufzeigen der wesentlichen Vorgehensweisen. Der Umfang des Stoffes orientiert sich daran, das RSA-Verfahren verstehen und anwenden zu können.

Dazu wird sowohl an Beispielen als auch in der Theorie erklärt, wie man in endlichen Mengen rechnet und wie dies in der Kryptographie Anwendung findet. Insbesondere wird auf die klassischen Public-Key-Verfahren Diffie-Hellman (DH) und RSA eingegangen.

Außerdem war es mir wichtig, fundierte Aussagen zur Sicherheit des RSA-Verfahrens zu machen.

*Carl Friedrich Gauss:*

Die Mathematik ist die Königin der Wissenschaften, die Zahlentheorie aber ist die Königin der Mathematik.

## 4.2 Einführung in die Zahlentheorie

Die Zahlentheorie entstand aus Interesse an den positiven ganzen Zahlen  $1, 2, 3, 4, \dots$ , die auch als die Menge der *natürlichen Zahlen*  $\mathbb{N}$  bezeichnet werden. Sie sind die ersten mathematischen

<sup>2</sup>Auf ggT, englisch gcd (greatest common divisor), geht dieser Artikel in Anhang 4.14 ein.

<sup>3</sup>Die Folge der Fibonacci-Zahlen  $(a_i)_{i \in \mathbb{N}}$  ist definiert durch die „rekursive“ Vorschrift  $a_1 := a_2 := 1$  und für alle Zahlen  $n = 1, 2, 3, \dots$  definiert man  $a_{n+2} := a_{n+1} + a_n$ . Zu dieser historischen Folge gibt es viele interessante Anwendungen in der Natur (siehe z.B. [Graham1994, S. 290 ff] oder die Web-Seite von Ron Knott: hier dreht sich alles um Fibonacci-Zahlen). Die Fibonacci-Folge ist gut verstanden und wird heute als wichtiges Werkzeug in der Mathematik benutzt.

Konstrukte der menschlichen Zivilisation. Nach Kronecker<sup>4</sup> hat sie der liebe Gott geschaffen, nach Dedekind<sup>5</sup> der menschliche Geist. Das ist je nach Weltanschauung ein unlösbarer Widerspruch oder ein und dasselbe.

Im Altertum gab es keinen Unterschied zwischen Zahlentheorie und Numerologie, die einzelnen Zahlen mystische Bedeutung zumaß. So wie sich während der Renaissance (ab dem 14. Jahrhundert) die Astronomie allmählich von der Astrologie und die Chemie von der Alchemie löste, so ließ auch die Zahlentheorie die Numerologie hinter sich.

Die Zahlentheorie faszinierte schon immer Amateure wie auch professionelle Mathematiker. Im Unterschied zu anderen Teilgebieten der Mathematik können viele der Probleme und Sätze auch von Laien verstanden werden, andererseits widersetzen sich die Lösungen zu den Problemen und die Beweise zu den Sätzen oft sehr lange den Mathematikern. Es ist also leicht, gute Fragen zu stellen, aber es ist ganz etwas anderes, die Antwort zu finden. Ein Beispiel dafür ist der sogenannte letzte (oder große) Satz von Fermat<sup>6</sup>.

Bis zur Mitte des 20. Jahrhunderts wurde die Zahlentheorie als das reinste Teilgebiet der Mathematik angesehen – ohne Verwendung in der wirklichen Welt. Mit dem Aufkommen der Computer und der digitalen Kommunikation änderte sich das: die Zahlentheorie konnte einige unerwartete Antworten für reale Aufgabenstellungen liefern. Gleichzeitig halfen die Fortschritte in der EDV, dass die Zahlentheoretiker große Fortschritte machten im Faktorisieren großer Zahlen, in der Bestimmung neuer Primzahlen, im Testen von (alten) Vermutungen und beim Lösen bisher unlösbarer numerischer Probleme.

Die moderne Zahlentheorie besteht aus Teilgebieten wie

- Elementare Zahlentheorie
- Algebraische Zahlentheorie
- Analytische Zahlentheorie
- Geometrische Zahlentheorie
- Kombinatorische Zahlentheorie
- Numerische Zahlentheorie und
- Wahrscheinlichkeitstheorie.

Die verschiedenen Teilgebiete beschäftigen sich alle mit Fragestellungen zu den ganzen Zahlen (positive und negative ganze Zahlen und die Null), gehen diese jedoch mit verschiedenen Methoden an.

Dieser Artikel beschäftigt sich nur mit dem Teilgebiet der elementaren Zahlentheorie.

---

<sup>4</sup>Leopold Kronecker, deutscher Mathematiker, 7.12.1823–29.12.1891.

<sup>5</sup>Julius Wilhelm Richard Dedekind, deutscher Mathematiker, 06.10.1831–12.02.1916.

<sup>6</sup>Thema der Schul-Mathematik ist der Satz von Pythagoras, wo gelehrt wird, dass in einem rechtwinkligen Dreieck gilt:  $a^2 + b^2 = c^2$ , wobei  $a, b$  die Schenkelängen sind und  $c$  die Länge der Hypotenuse ist. Fermats berühmte Behauptung war, dass für  $a, b, c \in \mathbb{N}$  und für ganzzahlige Exponenten  $n > 2$  immer die Ungleichheit  $a^n + b^n \neq c^n$  gilt. Leider fand Fermat auf dem Brief, wo er die Behauptung aufstellte, nicht genügend Platz, um den Satz zu beweisen. Der Satz konnte erst über 300 Jahre später bewiesen werden [Wiles1994, S. 433-551].

### 4.2.1 Konvention

Wird nichts anderes gesagt, gilt:

- Die Buchstaben  $a, b, c, d, e, k, n, m, p, q$  stehen für ganze Zahlen.
- Die Buchstaben  $i$  und  $j$  stehen für natürliche Zahlen.
- Der Buchstabe  $p$  steht stets für eine Primzahl.
- Die Mengen  $\mathbb{N} = \{1, 2, 3, \dots\}$  und  $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$  sind die *natürlichen* und die *ganzen* Zahlen.



Joanne K. Rowling<sup>7</sup>:

Das ist nicht Zauberei, das ist Logik, ein Rätsel. Viele von den größten Zauberern haben keine Unze Logik im Kopf.

### 4.3 Primzahlen und der erste Hauptsatz der elementaren Zahlentheorie

Viele der Probleme in der elementaren Zahlentheorie beschäftigen sich mit Primzahlen.

Jede ganze Zahl hat Teiler oder Faktoren. Die Zahl 1 hat nur einen, nämlich sich selbst. Die Zahl 12 hat die sechs Teiler 1, 2, 3, 4, 6 und 12<sup>8</sup>. Viele Zahlen sind nur teilbar durch sich selbst und durch 1. Bezüglich der Multiplikation sind dies die „Atome“ im Bereich der Zahlen.

**Definition 4.3.1. Primzahlen** *sind natürliche Zahlen größer als 1, die nur durch 1 und sich selbst teilbar sind.*

Per Definition ist 1 keine Primzahl.

Schreibt man die Primzahlen in aufsteigender Folge (Primzahlenfolge), so ergibt sich

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,  $\dots$ .

Unter den ersten 100 Zahlen gibt es genau 25 Primzahlen. Danach nimmt ihr prozentualer Anteil ab, wird aber nie Null.

Primzahlen treten als ganze Zahlen nicht selten auf. Allein im letzten Jahrzehnt waren drei Jahre prim: 1993, 1997 und 1999. Wären sie selten, könnte die Kryptographie auch nicht so mit ihnen arbeiten, wie sie es tut.

Primzahlen können nur auf eine einzige („triviale“) Weise zerlegt werden:

$$\begin{aligned}5 &= 1 * 5 \\17 &= 1 * 17 \\1.013 &= 1 * 1013 \\1.296.409 &= 1 * 1.296.409.\end{aligned}$$

**Definition 4.3.2. Natürliche Zahlen größer 1, die keine Primzahlen sind, heißen zusammengesetzte Zahlen:** *Diese haben mindestens zwei von 1 verschiedene Faktoren.*

<sup>7</sup>Joanne K. Rowling, „Harry Potter und der Stein der Weisen“, Carlsen, (c) 1997, Kapitel „Durch die Falltür“, S. 310, Hermine.

<sup>8</sup>Aufgrund der großen Teilerzahl von 12 findet sich diese Zahl – und Vielfache dieser Zahl – oft im alltäglichen wieder: Die 12 Stunden-Skala der Uhr, die 60 Minuten einer Stunde, die 360 Grad-Skala der Winkelmessung, usw. Teilt man diese Skalen in Bruchteile auf, so ergeben sich in vielen Fällen die Brüche als ganze Zahlen. Mit diesen kann man im Kopf einfacher rechnen als mit gebrochenen Zahlen.

### Beispiel Primfaktorzerlegung solcher Zahlen:

$$\begin{aligned}4 &= 2 * 2 \\6 &= 2 * 3 \\91 &= 7 * 13 \\161 &= 7 * 23 \\767 &= 13 * 59 \\1.029 &= 3 * 7^3 \\5.324 &= 22 * 11^3.\end{aligned}$$

**Satz 4.3.1.** *Jede zusammengesetzte Zahl  $a$  besitzt einen kleinsten Teiler größer als 1. Dieser Teiler ist eine Primzahl  $p$  und kleiner oder gleich der Quadratwurzel aus  $a$ .*

Aus den Primzahlen lassen sich alle ganzen Zahlen größer als 1 zusammensetzen – und das sogar in einer *eindeutigen* Weise.

Dies besagt der 1. *Hauptsatz der Zahlentheorie* (= Hauptsatz der elementaren Zahlentheorie = fundamental theorem of arithmetic = fundamental building block of all positive integers). Er wurde das erste Mal präzise von Carl Friedrich Gauss in seinen *Disquisitiones Arithmeticae* (1801) formuliert.

**Satz 4.3.2. Gauss 1801** *Jede natürliche Zahl  $a$  größer als 1 lässt sich als Produkt von Primzahlen schreiben. Sind zwei solche Zerlegungen  $a = p_1 * p_2 * \dots * p_n = q_1 * q_2 * \dots * q_m$  gegeben, dann gilt nach eventuellem Umsortieren  $n = m$  und für alle  $i$ :  $p_i = q_i$ .*

In anderen Worten: Jede natürliche Zahl außer der 1 lässt sich auf genau eine Weise als Produkt von Primzahlen schreiben, wenn man von der Reihenfolge der Faktoren absieht. Die Faktoren sind also eindeutig (die „Expansion in Faktoren“ ist eindeutig)!

Zum Beispiel ist  $60 = 2 * 2 * 3 * 5 = 2^2 * 3 * 5$ . Und das ist — bis auf eine veränderte Reihenfolge der Faktoren — die einzige Möglichkeit, die Zahl 60 in Primfaktoren zu zerlegen.

Wenn man nicht nur Primzahlen als Faktoren zulässt, gibt es mehrere Möglichkeiten der Zerlegung in Faktoren und die *Eindeutigkeit* (uniqueness) geht verloren:

$$60 = 1 * 60 = 2 * 30 = 4 * 15 = 5 * 12 = 6 * 10 = 2 * 3 * 10 = 2 * 5 * 6 = 3 * 4 * 5 = \dots$$

Der 1. Hauptsatz ist nur scheinbar selbstverständlich. Man kann viele andere Zahlenmengen<sup>9</sup> konstruieren, bei denen eine multiplikative Zerlegung in die Primfaktoren dieser Mengen *nicht* eindeutig ist.

Für eine mathematische Aussage ist es deshalb nicht nur wichtig, für welche Operation sie definiert wird, sondern auch auf welcher Grundmenge diese Operation definiert wird.

Weitere Details zu den Primzahlen (z.B. wie der „Kleine Satz von Fermat“ zum Testen von sehr großen Zahlen auf ihre Primzahleigenschaft benutzt werden kann) finden sich in diesem Skript in dem Artikel über Primzahlen, Kapitel 3.

---

<sup>9</sup>Diese Mengen werden speziell aus der Menge der natürlichen Zahlen gebildet. Ein Beispiel findet sich in diesem Skript auf Seite 54 am Ende von Kapitel 3.2.

## 4.4 Teilbarkeit, Modulus und Restklassen<sup>10</sup>

Werden ganze Zahlen addiert, subtrahiert oder multipliziert, ist das Ergebnis stets wieder eine ganze Zahl. Die Division zweier ganzer Zahlen ergibt nicht immer eine ganze Zahl. Wenn man z.B. 158 durch 10 teilt, ist das Ergebnis die Dezimalzahl 15,8. Dies ist keine ganze Zahl!

Teilt man 158 dagegen durch 2, ist das Ergebnis 79 eine ganze Zahl. In der Zahlentheorie sagt man, 158 ist *teilbar* durch 2, aber nicht durch 10. Allgemein sagt man:

**Definition 4.4.1.** Eine ganze Zahl  $n$  ist **teilbar** durch eine ganze Zahl  $d$ , wenn der Quotient  $n/d$  eine ganze Zahl  $c$  ist, so dass  $n = c * d$ .

Die Zahl  $n$  wird *Vielfaches* von  $d$  genannt;  $d$  wird *Teiler*, *Divisor* oder *Faktor* von  $n$  genannt.

Mathematisch schreibt man das:  $d|n$  (gelesen: „ $d$  teilt  $n$ “). Die Schreibweise  $d \nmid n$  bedeutet, dass  $d$  die Zahl  $n$  nicht teilt.

Also gilt in unserem obigen Beispiel:  $10 \nmid 158$ , aber  $2|158$ .

### 4.4.1 Die Modulo-Operation – Rechnen mit Kongruenzen

Bei Teilbarkeitsuntersuchungen kommt es nur auf die Reste der Division an: Teilt man eine Zahl  $n$  durch  $m$ , so benutzt man oft die folgende Schreibweise:

$$\frac{n}{m} = c + \frac{r}{m},$$

wobei  $c$  eine ganze Zahl ist und  $r$  eine Zahl mit den Werten  $0, 1, \dots, m-1$ . Diese Schreibweise heißt Division mit Rest. Dabei heißt  $c$  der ganzzahlige „Quotient“ und  $r$  der „Rest“ der Division.

**Beispiel:**

$$\frac{19}{7} = 2 + \frac{5}{7} \quad (m = 7, c = 2, r = 5)$$

Was haben die Zahlen 5, 12, 19, 26, ... bei der Division durch 7 gemeinsam? Es ergibt sich immer der Rest  $r = 5$ . Bei der Division durch 7 sind nur die folgenden Reste möglich:

$$r = 0, 1, 2, \dots, 6.$$

Wir fassen bei der Division durch 7 die Zahlen, die den gleichen Rest  $r$  ergeben, in die „Restklasse  $r$  modulo 7“ zusammen. Zwei Zahlen  $a$  und  $b$ , die zur gleichen Restklasse modulo 7 gehören, bezeichnen wir als „kongruent modulo 7“. Oder ganz allgemein:

**Definition 4.4.2.** Als **Restklasse  $r$  modulo  $m$**  bezeichnet man alle ganzen Zahlen  $a$ , die bei der Division durch  $m$  denselben Rest  $r$  haben.

<sup>10</sup>Mit dem Lernprogramm **ZT** können Sie das hier und im Folgekapitel vorgestellte Rechnen mit Kongruenzen spielerisch nachvollziehen (siehe Lern-Kapitel 2.1, Seiten 2-9/40).

ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

**Beispiel:**

Restklasse 0 modulo 4 =  $\{x|x = 4 * n; n \in \mathbb{Z}\} = \{\dots, -16, -12, -8, -4, 0, 4, 8, 12, 16, \dots\}$

Restklasse 3 modulo 4 =  $\{x|x = 4 * n + 3; n \in \mathbb{Z}\} = \{\dots, -13, -9, -5, -1, 3, 7, 11, 15, \dots\}$

Da modulo  $m$  nur die Reste  $0, 1, 2, \dots, m-1$  möglich sind, rechnet die modulare Arithmetik in endlichen Mengen. Zu jedem Modul  $m$  gibt es genau  $m$  Restklassen.

**Definition 4.4.3.** Zwei Zahlen  $a, b \in \mathbb{N}$  heißen **restgleich oder kongruent bezüglich**  $m \in \mathbb{N}$  genau dann, wenn beim Teilen durch  $m$  der gleiche Rest bleibt.

Man schreibt:  $a \equiv b \pmod{m}$ . Und sagt: *a ist kongruent b modulo m*. Das bedeutet, dass  $a$  und  $b$  zur gleichen Restklasse gehören. Der Modul ist also der Teiler. Diese Schreibweise wurde von Gauss eingeführt. Gewöhnlich ist der Teiler positiv, aber  $a$  und  $b$  können auch beliebige ganze Zahlen sein.

**Beispiel:**

$19 \equiv 12 \pmod{7}$ , denn die Reste sind gleich:  $19/7 = 2$  Rest 5 und  $12/7 = 1$  Rest 5.

$23103 \equiv 0 \pmod{453}$ , denn  $23103/453 = 51$  Rest 0 und  $0/453 = 0$  Rest 0.

**Satz 4.4.1.**  $a \equiv b \pmod{m}$  gilt genau dann, wenn die Differenz  $(a - b)$  durch  $m$  teilbar ist, also wenn ein  $q \in \mathbb{Z}$  existiert mit  $(a - b) = q * m$ .

Diese beiden Aussagen sind also äquivalent.

Daraus ergibt sich: Wenn  $m$  die Differenz teilt, gibt es eine ganze Zahl  $q$ , so dass gilt:  $a = b + q * m$ . Alternativ zur Kongruenzschreibweise kann man auch die Teilbarkeitsschreibweise verwenden:  $m|(a - b)$ .

**Beispiel für äquivalente Aussagen:**

$35 \equiv 11 \pmod{3} \iff 35 - 11 \equiv 0 \pmod{3}$ , wobei  $35 - 11 = 24$  sich ohne Rest durch 3 teilen lässt, während  $35 : 3$  und  $11 : 3$  beide den Rest 2 ergeben.

**Bemerkung:**

Für die Summe  $(a + b)$  gilt die obige Äquivalenz nicht!

**Beispiel:**

$11 \equiv 2 \pmod{3}$ , also ist  $11 - 2 = 9 \equiv 0 \pmod{3}$ ; aber  $11 + 2 = 13$  ist nicht durch 3 teilbar.

Die Aussage von Satz 4.4.1 gilt für Summen nicht einmal in eine Richtung. Richtig ist sie bei Summen nur für den Rest 0 und nur in der folgenden Richtung: Teilt ein Teiler beide Summanden ohne Rest, teilt er auch die Summe ohne Rest.

Anwenden kann man die obige Äquivalenz von Satz 4.4.1, wenn man schnell und geschickt für große Zahlen entscheiden will, ob sie durch eine bestimmte Zahl teilbar sind.

**Beispiel:**

Ist 69.993 durch 7 teilbar?

Da die Zahl in eine Differenz zerlegt werden kann, wo einfach zu erkennen ist, dass jeder Operand durch 7 teilbar ist, ist auch die Differenz durch 7 teilbar:  $69.993 = 70.000 - 7$ .

Diese Überlegungen und Definitionen mögen recht theoretisch erscheinen, sind uns im Alltag aber so vertraut, dass wir die formale Vorgehensweise gar nicht mehr wahrnehmen: Bei der Uhr werden die 24 h eines Tages durch die Zahlen  $1, 2, \dots, 12$  repräsentiert. Die Stunden nach 12 : 00 mittags erhält man als Reste einer Division durch 12. Wir wissen sofort, dass 2 Uhr nachmittags dasselbe wie 14 : 00 ist.

Diese „modulare“, also auf die Divisionsreste bezogene Arithmetik ist die Basis der asymmetrischen Verschlüsselungsverfahren. Kryptographische Berechnungen spielen sich also nicht wie das Schulrechnen unter den reellen Zahlen ab, sondern unter Zeichenketten begrenzter Länge, das heißt unter positiven ganzen Zahlen, die einen gewissen Wert nicht überschreiten dürfen. Aus diesem und anderen Gründen wählt man sich eine große Zahl  $m$  und „rechnet modulo  $m$ “, das heißt, man ignoriert ganzzahlige Vielfache von  $m$  und rechnet statt mit einer Zahl nur mit dem Rest bei Division dieser Zahl durch  $m$ . Dadurch bleiben alle Ergebnisse im Bereich von 0 bis  $m - 1$ .

## 4.5 Rechnen in endlichen Mengen

### 4.5.1 Gesetze beim modularen Rechnen

Aus Sätzen der Algebra folgt, dass wesentliche Teile der üblichen Rechenregeln beim Übergang zum modularen Rechnen über der Grundmenge  $\mathbb{Z}$  erhalten bleiben: Die Addition ist nach wie vor kommutativ. Gleiches gilt für die Multiplikation modulo  $m$ . Das Ergebnis einer Division<sup>11</sup> ist kein Bruch, sondern eine ganze Zahl zwischen 0 und  $m - 1$ .

Es gelten die bekannten Gesetze:

**1. Assoziativgesetz:**

$$((a + b) + c) \pmod{m} \equiv (a + (b + c)) \pmod{m}.$$

$$((a * b) * c) \pmod{m} \equiv (a * (b * c)) \pmod{m}.$$

**2. Kommutativgesetz:**

$$(a + b) \pmod{m} \equiv (b + a) \pmod{m}.$$

$$(a * b) \pmod{m} \equiv (b * a) \pmod{m}.$$

Assoziativgesetz und Kommutativgesetz gelten sowohl für die Addition als auch für die Multiplikation.

**3. Distributivgesetz:**

$$(a * (b + c)) \pmod{m} \equiv (a * b + a * c) \pmod{m}.$$

**4. Reduzierbarkeit:**

$$(a + b) \pmod{m} \equiv (a \pmod{m} + b \pmod{m}) \pmod{m}.$$

$$(a * b) \pmod{m} \equiv (a \pmod{m} * b \pmod{m}) \pmod{m}.$$

Beim Addieren und Multiplizieren ist es gleichgültig, in welcher Reihenfolge die Modulo-Operation durchgeführt wird.

---

<sup>11</sup>Die Division modulo  $m$  ist nur für Zahlen, die teilerfremd zu  $m$  sind, definiert, da andere Zahlen die gleiche Eigenschaft wie Null haben. Vergleiche Fußnote 15 in Kapitel 4.6.1.

**5. Existenz einer Identität (neutrales Element):**

$$(a + 0) \pmod{m} \equiv (0 + a) \pmod{m} \equiv a \pmod{m}.$$

$$(a * 1) \pmod{m} \equiv (1 * a) \pmod{m} \equiv a \pmod{m}.$$

**6. Existenz des inversen Elements:**

Für jedes ganzzahlige  $a$  und  $m$  gibt es eine ganze Zahl  $-a$ , so dass gilt:

$$(a + (-a)) \pmod{m} \equiv 0 \pmod{m} \quad (\text{additive Inverse}).$$

Für jedes  $a$  ( $a \not\equiv 0 \pmod{p}$ ) und  $p$  prim gibt es eine ganze Zahl  $a^{-1}$ , so dass gilt:

$$(a * a^{-1}) \pmod{p} \equiv 1 \pmod{p} \quad (\text{multiplikative Inverse}).$$

**7. Abgeschlossenheit:**<sup>12</sup>

$$a, b \in G \implies (a + b) \in G.$$

$$a, b \in G \implies (a * b) \in G.$$

**8. Transitivität:**

$$[a \equiv b \pmod{m}, b \equiv c \pmod{m}] \implies [a \equiv c \pmod{m}].$$

### 4.5.2 Muster und Strukturen

Generell untersuchen die Mathematiker „Strukturen“. Sie fragen sich z.B. bei  $a * x \equiv b \pmod{m}$ , welche Werte  $x$  für gegebene Werte  $a, b, m$  annehmen kann.

Insbesondere wird dies untersucht für den Fall, dass das Ergebnis  $b$  der Operation das neutrale Element ist. Dann ist  $x$  die Inverse von  $a$  bezüglich dieser Operation.

---

<sup>12</sup>Diese Eigenschaft wird innerhalb einer Menge immer bezüglich einer Operation definiert. Siehe Kapitel 4.15 „Anhang: Abschlussbildung“.

*Seneca*<sup>13</sup>:

Lang ist der Weg durch Lehren, kurz und wirksam durch Beispiele.

## 4.6 Beispiele für modulares Rechnen

Wir haben bisher gesehen:

Für zwei natürliche Zahlen  $a$  und  $m$  bezeichnet  $a \bmod m$  den Rest, den man erhält, wenn man  $a$  durch  $m$  teilt. Daher ist  $a \bmod m$  stets eine Zahl zwischen 0 und  $m - 1$ .

Zum Beispiel gilt:  $1 \equiv 6 \equiv 41 \pmod{5}$ , denn der Rest ist jeweils 1.

Ein anderes Beispiel ist:  $2000 \equiv 0 \pmod{4}$ , denn 4 geht in 2000 ohne Rest auf.

In der modularen Arithmetik gibt es nur eine eingeschränkte Menge nicht-negativer Zahlen. Deren Anzahl wird durch einen Modul  $m$  vorgegeben. Ist der Modul  $m = 5$ , werden nur die 5 Zahlen der Menge  $\{0, 1, 2, 3, 4\}$  benutzt.

Ein Rechenergebnis größer als 4 wird dann „modulo“ 5 umgeformt, d.h. es ist der Rest, der sich bei der Division des Ergebnisses durch 5 ergibt. So ist etwa  $2 * 4 \equiv 8 \equiv 3 \pmod{5}$ , da 3 der Rest ist, wenn man 8 durch 5 teilt.

### 4.6.1 Addition und Multiplikation

Im folgenden werden

- die Additionstabelle<sup>14</sup> für mod 5 (Tabelle 4.1) und
- die Multiplikationstabellen<sup>15</sup> für mod 5 (Tabelle 4.2) und für mod 6 (Tabelle 4.3)

aufgestellt.

#### Beispiel Additionstabelle:

Das Ergebnis der Addition von 3 und 4 (mod 5) wird folgendermaßen bestimmt: berechne  $3+4 = 7$  und ziehe solange die 5 vom Ergebnis ab, bis sich ein Ergebnis kleiner als der Modul ergibt:  $7 - 5 = 2$ . Also ist:  $3 + 4 \equiv 2 \pmod{5}$ .

**Beispiel Multiplikationstabelle:** Das Ergebnis der Multiplikation  $4 * 4 \pmod{5}$  wird folgendermaßen berechnet: berechne  $4 * 4 = 16$  und ziehe solange die 5 ab, bis sich ein Ergebnis kleiner als der Modul ergibt:

$$16 - 5 = 11; 11 - 5 = 6; 6 - 5 = 1.$$

Direkt ergibt es sich auch aus Tabelle 4.2:  $4 * 4 \equiv 1 \pmod{5}$ , weil  $16 : 5 = 3$  Rest 1. Die Multiplikation wird auf der Menge  $\mathbb{Z}$  ohne 0 definiert.

<sup>13</sup>Lucius Annaeus Seneca, philosophischer Schriftsteller und Dichter, 4 v. Chr. – 65 n. Chr.

<sup>14</sup>Bemerkung zur Subtraktion modulo 5:

$$2 - 4 = -2 \equiv 3 \pmod{5}.$$

Es gilt modulo 5 also nicht, dass  $-2 = 2$  (siehe Kapitel 4.16 “Anhang: Bemerkungen zur modulo Subtraktion”).

<sup>15</sup>Bemerkung zur Division modulo 6:

Bei der Division darf nicht durch die Null geteilt werden, dies liegt an der besonderen Rolle der 0 als Identität bei der Addition:

Für alle  $a$  gilt  $a * 0 = 0$ , denn  $a * 0 = a * (0 + 0) = a * 0 + a * 0$ . Es ist offensichtlich, dass 0 keine Inverse bzgl. der Multiplikation besitzt, denn sonst müsste gelten:  $0 = 0 * 0^{-1} = 1$ . Vergleiche Fußnote 11.

+	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 4.1: Additionstabelle modulo 5

*	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 4.2: Multiplikationstabelle modulo 5

#### 4.6.2 Additive und multiplikative Inversen

Aus den Tabellen kann man zu jeder Zahl die Inversen bezüglich der Addition und der Multiplikation ablesen.

Die Inverse einer Zahl ist diejenige Zahl, die bei Addition der beiden Zahlen das Ergebnis 0 und bei der Multiplikation das Ergebnis 1 ergibt. So ist die Inverse von 4 für die Addition mod 5 die 1 und für die Multiplikation mod 5 die 4 selbst, denn

$$\begin{aligned}4 + 1 &= 5 \equiv 0 \pmod{5}; \\4 * 4 &= 16 \equiv 1 \pmod{5}.\end{aligned}$$

Die Inverse von 1 bei der Multiplikation mod 5 ist 1; die Inverse modulo 5 von 2 ist 3 und weil die Multiplikation kommutativ ist, ist die Inverse von 3 wiederum die 2.

Wenn man zu einer beliebigen Zahl (hier 2) eine weitere beliebige Zahl (hier 4) addiert bzw. multipliziert und danach zum Zwischenergebnis (1 bzw. 3) die jeweilige Inverse der weiteren Zahl (1 bzw. 4) addiert<sup>16</sup> bzw. multipliziert, ist das Gesamtergebnis gleich dem Ausgangswert.

**Beispiel:**

$$\begin{aligned}2 + 4 &\equiv 6 \equiv 1 \pmod{5}; & 1 + 1 &\equiv 2 \equiv 2 \pmod{5} \\2 * 4 &\equiv 8 \equiv 3 \pmod{5}; & 3 * 4 &\equiv 12 \equiv 2 \pmod{5}\end{aligned}$$

In der Menge  $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$  für die Addition und in der Menge  $\mathbb{Z}_5 \setminus \{0\}$  für die Multiplikation haben alle Zahlen eine **eindeutige** Inverse bezüglich modulo 5.

Bei der modularen Addition ist das für jeden Modul (also nicht nur für 5) so.

Bei der modularen Multiplikation dagegen ist das nicht so:

<sup>16</sup>Allgemein:  $x + y + (-y) \equiv x \pmod{m}$  [ $(-y)$  = additive Inverse zu  $y \pmod{m}$ ]



**Satz 4.6.1.** Für eine natürliche Zahl  $a$  aus der Menge  $\{1, \dots, m-1\}$  gibt es genau dann eine multiplikative Inverse, wenn sie mit dem Modul  $m$  teilerfremd<sup>17</sup> ist, d.h. wenn  $a$  und  $m$  keine gemeinsamen Primfaktoren haben.

Da  $m = 5$  eine Primzahl ist, sind die Zahlen 1 bis 4 teilerfremd zu 5, und es gibt mod 5 zu **jeder** dieser Zahlen eine multiplikative Inverse.

Ein Gegenbeispiel zeigt die Multiplikationstabelle für mod 6 (da der Modul  $m = 6$  nicht prim ist, sind nicht alle Elemente aus  $\mathbb{Z}_6 \setminus \{0\}$  zu 6 teilerfremd):

*	1	2	3	4	5
1	1	2	3	4	5
2	2	<b>4</b>	<b>0</b>	<b>2</b>	4
3	3	<b>0</b>	<b>3</b>	<b>0</b>	3
4	4	<b>2</b>	<b>0</b>	<b>4</b>	2
5	5	4	3	2	1

Tabelle 4.3: Multiplikationstabelle modulo 6

Neben der 0 haben hier auch die Zahlen 2, 3 und 4 keine eindeutige Inverse (man sagt auch, sie haben **keine** Inverse, weil es die elementare Eigenschaft einer Inversen ist, eindeutig zu sein).

Die Zahlen 2, 3 und 4 haben mit dem Modul 6 den Faktor 2 oder 3 gemeinsam. Nur die zu 6 teilerfremden Zahlen 1 und 5 haben multiplikative Inverse, nämlich jeweils sich selbst.

Die Anzahl der zum Modul  $m$  teilerfremden Zahlen ist auch die Anzahl derjenigen Zahlen, die eine multiplikative Inverse haben (vgl. unten die Euler-Funktion  $J(m)$ ).

Für die beiden in den Multiplikationstabellen verwendeten Moduli 5 und 6 bedeutet dies: Der Modul 5 ist bereits eine Primzahl. Also gibt es in mod 5 genau  $J(5) = 5 - 1 = 4$  mit dem Modul teilerfremde Zahlen, also alle von 1 bis 4.

Da 6 keine Primzahl ist, zerlegen wir 6 in seine Faktoren:  $6 = 2 * 3$ . Daher gibt es in mod 6 genau  $J(6) = (2 - 1) * (3 - 1) = 1 * 2 = 2$  Zahlen, die eine multiplikative Inverse haben, nämlich die 1 und die 5.

Für große Moduli scheint es nicht einfach, die Tabelle der multiplikativen Inversen zu berechnen (das gilt nur für die in den oberen Multiplikationstabellen fett markierten Zahlen). Mit Hilfe des kleinen Satzes von Fermat kann man dafür einen einfachen Algorithmus aufstellen [Pfleeger1997, S. 80]. Schnellere Algorithmen werden z.B. in [Knuth1998] beschrieben<sup>18</sup>.

Kryptographisch ist nicht nur die Eindeutigkeit der Inversen, sondern auch das Ausschöpfen

<sup>17</sup>Es gilt: Zwei ganze Zahlen  $a$  und  $b$  sind genau dann teilerfremd, wenn  $\text{ggT}(a, b) = 1$ .

Desweiteren gilt: Ist  $p$  prim und  $a$  eine beliebige ganze Zahl, die kein Vielfaches von  $p$  ist, so sind beide Zahlen teilerfremd.

Weitere Bezeichnungen zum Thema Teilerfremdheit (mit  $a_i \in \mathbb{Z}, i = 1, \dots, n$ ):

1.  $a_1, a_2, \dots, a_n$  heißen *relativ prim*, wenn  $\text{ggT}(a_1, \dots, a_n) = 1$ .
2. Für mehr als 2 Zahlen ist eine noch stärkere Anforderung:  
 $a_1, \dots, a_n$  heißen *paarweise relativ prim*, wenn für alle  $i = 1, \dots, n$  und  $j = 1, \dots, n$  mit  $i \neq j$  gilt:  
 $\text{ggT}(a_i, a_j) = 1$ .

**Beispiel:**

2, 3, 6 sind relativ prim, da  $\text{ggT}(2, 3, 6) = 1$ . Sie sind nicht paarweise prim, da  $\text{ggT}(2, 6) = 2 > 1$ .

<sup>18</sup>Mit dem erweiterten Satz von Euklid (erweiterter ggT) kann man die multiplikative Inverse berechnen und die Invertierbarkeit bestimmen (siehe Anhang 4.14). Alternativ kann auch die Primitivwurzel genutzt werden.

des gesamten Wertebereiches eine wichtige Eigenschaft.

**Satz 4.6.2.** Sei  $a, i \in \{1, \dots, m-1\}$  mit  $\text{ggT}(a, m) = 1$ , dann nimmt für eine bestimmte Zahl  $a$  das Produkt  $a * i \bmod m$  alle Werte aus  $\{1, \dots, m-1\}$  an (erschöpfende Permutation der Länge  $m-1$ )<sup>19</sup>.

Die folgenden drei Beispiele<sup>20</sup> veranschaulichen Eigenschaften der multiplikativen Inversen (hier sind nicht mehr die vollständigen Multiplikationstabellen angegeben, sondern nur die Zeilen für die Faktoren 5 und 6).

In der Multiplikationstabelle mod 17 (Tabelle 4.4) wurde für  $i = 1, 2, \dots, 18$  berechnet:

$$(5 * i)/17 = a \text{ Rest } r \text{ und hervorgehoben } 5 * i \equiv 1 \pmod{17},$$

$$(6 * i)/17 = a \text{ Rest } r \text{ und hervorgehoben } 6 * i \equiv 1 \pmod{17}.$$

**Gesucht** ist das  $i$ , für das der Produktrest  $a * i$  modulo 17 mit  $a = 5$  bzw.  $a = 6$  den Wert 1 hat.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$5 * i$	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Rest	5	10	15	3	8	13	<b>1</b>	6	11	16	4	9	14	2	7	12	0	5
$6 * i$	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
Rest	6	12	<b>1</b>	7	13	2	8	14	3	9	15	4	10	16	5	11	0	6

Tabelle 4.4: Multiplikationstabelle modulo 17 (für  $a = 5$  und  $a = 6$ )

Da sowohl 5 als auch 6 jeweils teilerfremd zum Modul  $m = 17$  sind, kommen zwischen  $i = 1, \dots, m$  für die Reste alle Werte zwischen  $0, \dots, m-1$  vor (vollständige  $m$ -Permutation).

**Die multiplikative Inverse von 5 (mod 17) ist 7, die Inverse von 6 (mod 17) ist 3.**

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$5 * i$	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Rest	5	10	2	7	12	4	9	<b>1</b>	6	11	3	8	0	5	10	2	7	12
$6 * i$	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
Rest	6	12	5	11	4	10	3	9	2	8	<b>1</b>	7	0	6	12	5	11	4

Tabelle 4.5: Multiplikationstabelle modulo 13 (für  $a = 5$  und  $a = 6$ )

Da sowohl 5 als auch 6 auch zum Modul  $m = 13$  jeweils teilerfremd sind, kommen zwischen  $i = 1, \dots, m$  für die Reste alle Werte zwischen  $0, \dots, m-1$  vor.

**Die multiplikative Inverse von 5 (mod 13) ist 8, die Inverse von 6 (mod 13) ist 11.**

Tabelle 4.6 enthält ein Beispiel dafür, wo der Modul  $m$  und die Zahl ( $a = 6$ ) *nicht* teilerfremd sind.

Berechnet wurde  $(5 * i) \bmod 12$  und  $(6 * i) \bmod 12$ . Da 6 und der Modul  $m = 12$  nicht teilerfremd sind, kommen zwischen  $i = 1, \dots, m$  nicht alle Werte zwischen  $0, \dots, m-1$  vor und 6 hat mod 12 auch keine Inverse!

<sup>19</sup>Vergleiche auch Satz 4.9.1 in Kapitel 4.9, Multiplikative Ordnung und Primitivwurzel.

<sup>20</sup>In Kapitel 4.18.1 "Multiplikationstabellen modulo m" finden Sie den Quellcode zur Berechnung der Tabellen mit Sage.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
5*i	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Rest	5	10	3	8	1	6	11	4	9	2	7	0	5	10	3	8	1	6
6*i	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
Rest	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0

Tabelle 4.6: Multiplikationstabelle modulo 12 (für  $a = 5$  und  $a = 6$ )

**Die multiplikative Inverse von 5 (mod 12) ist 5. Die Zahl 6 hat keine Inverse (mod 12).**

### 4.6.3 Potenzieren

Das Potenzieren ist in der modularen Arithmetik definiert als wiederholtes Multiplizieren – wie üblich, nur dass jetzt Multiplizieren etwas anderes ist. Es gelten mit kleinen Einschränkungen die üblichen Rechenregeln wie

$$\begin{aligned} a^{b+c} &= a^b * a^c, \\ (a^b)^c &= a^{b*c} = a^{c*b} = (a^c)^b. \end{aligned}$$

Analog der modularen Addition und der modularen Multiplikation funktioniert das modulare Potenzieren:

$$3^2 = 9 \equiv 4 \pmod{5}.$$

Auch aufeinanderfolgendes Potenzieren geht analog:

**Beispiel 1:**

$$(4^3)^2 = 64^2 \equiv 4096 \equiv 1 \pmod{5}.$$

(1) Reduziert man bereits **Zwischenergebnisse** modulo 5, kann man schneller<sup>21</sup> rechnen, muss aber auch aufpassen, da sich dann *nicht* immer alles wie in der gewöhnlichen Arithmetik verhält.

$$\begin{aligned} (4^3)^2 &\equiv (4^3 \pmod{5})^2 \pmod{5} \\ &\equiv (64 \pmod{5})^2 \pmod{5} \\ &\equiv 4^2 \pmod{5} \\ &\equiv 16 \equiv 1 \pmod{5}. \end{aligned}$$

(2) Aufeinanderfolgende Potenzierungen lassen sich in der gewöhnlichen Arithmetik auf eine einfache Potenzierung zurückführen, indem man die Exponenten miteinander multipliziert:

$$(4^3)^2 = 4^{3*2} = 4^6 = 4096.$$

In der modularen Arithmetik geht das nicht ganz so einfach, denn man erhielte:

$$(4^3)^2 \equiv 4^{3*2 \pmod{5}} \equiv 4^6 \pmod{5} \equiv 4^1 \equiv 4 \pmod{5}.$$

<sup>21</sup>Die Rechenzeit der Multiplikation zweier Zahlen hängt normalerweise von der Länge der Zahlen ab. Dies sieht man, wenn man nach der Schulmethode z.B.  $474 * 228$  berechnet: Der Aufwand steigt quadratisch, da  $3 * 3$  Ziffern multipliziert werden müssen. Durch die Reduktion der Zwischenergebnisse werden die Zahlen deutlich kleiner.

Wie wir oben sahen, ist das richtige Ergebnis aber 1 !!

(3) Deshalb ist für das fortgesetzte Potenzieren in der modularen Arithmetik die Regel etwas anders: Man multipliziert die Exponenten nicht in  $(\text{mod } m)$ , sondern in  $(\text{mod } J(m))$ .

Mit  $J(5) = 4$  ergibt sich:

$$(4^3)^2 \equiv 4^{3*2 \pmod{J(5)}} \equiv 4^{6 \pmod{4}} \equiv 4^2 \equiv 16 \equiv 1 \pmod{5}.$$

Das liefert das richtige Ergebnis.

**Satz 4.6.3.**  $(a^b)^c \equiv a^{b*c \pmod{J(m)}} \pmod{m}$ .

**Beispiel 2:**

$$3^{28} = 3^{4*7} \equiv 3^{4*7 \pmod{10}} \equiv 3^8 \equiv 6561 \equiv 5 \pmod{11}.$$

#### 4.6.4 Schnelles Berechnen hoher Potenzen

Bei RSA-Ver- und Entschlüsselungen<sup>22</sup> müssen hohe Potenzen modulo  $m$  berechnet werden. Schon die Berechnung  $(100^5) \pmod{3}$  sprengt den 32 Bit langen Long-Integer-Zahlenbereich, sofern man zur Berechnung von  $a^n$  getreu der Definition  $a$  tatsächlich  $n$ -mal mit sich selbst multipliziert. Bei sehr großen Zahlen wäre selbst ein schneller Computerchip mit einer einzigen Exponentiation länger beschäftigt als das Weltall existiert. Glücklicherweise gibt es für die Exponentiation (nicht aber für das Logarithmieren) eine sehr wirksame Abkürzung.

Wird der Ausdruck anhand der Rechenregeln der modularen Arithmetik anders aufgeteilt, sprengt die Berechnung nicht einmal den 16 Bit langen Short-Integer-Bereich:

$$(a^5) \equiv (((a^2 \pmod{m})^2 \pmod{m}) * a) \pmod{m}.$$

Dies kann man verallgemeinern, indem man den Exponenten binär darstellt. Beispielsweise würde die Berechnung von  $a^n$  für  $n = 37$  auf dem naiven Wege 36 Multiplikationen erfordern. Schreibt man jedoch  $n$  in der Binärdarstellung als  $100101 = 1 * 2^5 + 1 * 2^2 + 1 * 2^0$ , so kann man umformen:  $a^{37} = a^{2^5+2^2+2^0} = a^{2^5} * a^{2^2} * a^{2^0}$ .

**Beispiel 3:**  $87^{43} \pmod{103}$ .

Da  $43 = 32 + 8 + 2 + 1$ , 103 prim,  $43 < J(103)$  ist und die Quadrate  $(\text{mod } 103)$  vorab berechnet werden können

$$\begin{aligned} 87^2 &\equiv 50 \pmod{103}, \\ 87^4 &\equiv 50^2 \equiv 28 \pmod{103}, \\ 87^8 &\equiv 28^2 \equiv 63 \pmod{103}, \\ 87^{16} &\equiv 63^2 \equiv 55 \pmod{103}, \\ 87^{32} &\equiv 55^2 \equiv 38 \pmod{103}, \end{aligned}$$

<sup>22</sup>Siehe Kapitel 4.10 (Beweis des RSA-Verfahrens mit Euler-Fermat) und Kapitel 4.13 (Das RSA-Verfahren mit konkreten Zahlen).

gilt<sup>23</sup>:

$$\begin{aligned} 87^{43} &\equiv 87^{32+8+2+1} \pmod{103} \\ &\equiv 87^{32} * 87^8 * 87^2 * 87 \pmod{103} \\ &\equiv 38 * 63 * 50 * 87 \equiv 85 \pmod{103}. \end{aligned}$$

Die Potenzen  $(a^2)^k$  sind durch fortgesetztes Quadrieren leicht zu bestimmen. Solange sich  $a$  nicht ändert, kann ein Computer sie vorab berechnen und – bei ausreichend Speicherplatz – abspeichern. Um dann im Einzelfall  $a^n$  zu finden, muss er nur noch genau diejenigen  $(a^2)^k$  miteinander multiplizieren, für die an der  $k$ -ten Stelle der Binärdarstellung von  $n$  eine Eins steht. Der typische Aufwand für  $n = 600$  sinkt dadurch von  $2^{600}$  auf  $2 * 600$  Multiplikationen! Dieser häufig verwendete Algorithmus heißt „Square and Multiply“.

#### 4.6.5 Wurzeln und Logarithmen

Die Umkehrungen des Potenzierens sind ebenfalls definiert: Wurzeln und Logarithmen sind abermals ganze Zahlen, aber im Gegensatz zur üblichen Situation sind sie nicht nur mühsam, sondern bei sehr großen Zahlen in „erträglicher“ Zeit überhaupt nicht zu berechnen.

Gegeben sei die Gleichung:  $a \equiv b^c \pmod{m}$ .

##### a) Logarithmieren (Bestimmen von $c$ ) – Diskretes Logarithmusproblem:<sup>24</sup>

Wenn man von den drei Zahlen  $a, b, c$ , welche diese Gleichung erfüllen,  $a$  und  $b$  kennt, ist jede bekannte Methode,  $c$  zu finden, ungefähr so aufwendig wie das Durchprobieren aller  $m$  denkbaren Werte für  $c$  – bei einem typischen  $m$  in der Größenordnung von  $10^{180}$  für 600-stellige Binärzahlen ein hoffnungsloses Unterfangen. Genauer ist für geeignet große Zahlen  $m$  der Aufwand nach heutigem Wissensstand proportional zu

$$\exp\left(C * (\log m [\log \log m]^2)^{1/3}\right)$$

mit einer Konstanten  $C > 1$ .

##### b) Wurzel-Berechnung (Bestimmen von $b$ ):

Ähnliches gilt, wenn  $b$  die Unbekannte ist und die Zahlen  $a$  und  $c$  bekannt sind.

Wenn die Eulerfunktion  $J(m)$  bekannt ist, findet man leicht<sup>25</sup>  $d$  mit  $c * d \equiv 1 \pmod{J(m)}$  und erhält mit Satz 4.6.3

$$a^d \equiv (b^c)^d \equiv b^{c*d} \equiv b^{c*d \pmod{J(m)}} \equiv b^1 \equiv b \pmod{m}$$

die  $c$ -te Wurzel  $b$  von  $a$ .

Für den Fall, dass  $J(m)$  in der Praxis nicht bestimmt werden kann<sup>26</sup>, ist die Berechnung der  $c$ -ten Wurzel schwierig. Hierauf beruhen die Sicherheitsannahmen für das RSA-Kryptosystem (siehe Kapitel 4.10 oder Kapitel 5.3.1).

<sup>23</sup>In Kapitel 4.18.2 „Schnelles Berechnen hoher Potenzen“ finden Sie den Beispielcode zum Nachrechnen der Square-and-Multiply Methode mit Sage.

<sup>24</sup>Weitere Details zum Diskreten Logarithmusproblem finden sie in Kapitel 5.4.

<sup>25</sup>Siehe Kapitel 4.14 „Anhang: Der ggT und die beiden Algorithmen von Euklid“.

<sup>26</sup>Nach dem ersten Hauptsatz der Zahlentheorie und Satz 4.8.4 kann man  $J(m)$  mit Hilfe der Primfaktorzerlegung von  $m$  bestimmen.

Dagegen ist der Aufwand für die Umkehrung von Addition und Multiplikation nur proportional zu  $\log m$  beziehungsweise  $(\log m)^2$ . Potenzieren (zu einer Zahl  $x$  berechne  $x^a$  mit festem  $a$ ) und Exponentiation (zu einer Zahl  $x$  berechne  $a^x$  mit festem  $a$ ) sind also typische Einwegfunktionen (vergleiche Kapitel 5.1 und 4.12.1).

## 4.7 Gruppen und modulare Arithmetik über $\mathbb{Z}_n$ und $\mathbb{Z}_n^*$

In der Zahlentheorie und in der Kryptographie spielen mathematische „*Gruppen*“ eine entscheidende Rolle. Von Gruppen spricht man nur, wenn für eine definierte Menge und eine definierte Relation (eine Operation wie Addition oder Multiplikation) die folgenden Eigenschaften erfüllt sind:

- Abgeschlossenheit
- Existenz des neutralen Elements
- Existenz eines inversen Elements zu jedem Element und
- Gültigkeit des Assoziativgesetzes.

Die abgekürzte mathematische Schreibweise lautet:  $(G, +)$  oder  $(G, *)$ .

**Definition 4.7.1.**  $\mathbb{Z}_n$  :

$\mathbb{Z}_n$  umfasst alle ganzen Zahlen von 0 bis  $n - 1$  :  $\mathbb{Z}_n = \{0, 1, 2, \dots, n - 2, n - 1\}$ .

$\mathbb{Z}_n$  ist eine häufig verwendete endliche Gruppe aus den natürlichen Zahlen. Sie wird manchmal auch als Restemenge  $R$  modulo  $n$  bezeichnet.

Beispielsweise rechnen 32 Bit-Computer (übliche PCs) mit ganzen Zahlen direkt nur in einer endlichen Menge, nämlich in dem Wertebereich  $0, 1, 2, \dots, 2^{32} - 1$ .

Dieser Zahlenbereich ist äquivalent zur Menge  $\mathbb{Z}_{2^{32}}$ .

### 4.7.1 Addition in einer Gruppe

Definiert man auf einer solchen Menge die Operation  $\text{mod}+$  mit

$$a \text{ mod}+ b := (a + b) \pmod{n},$$

so ist die Menge  $\mathbb{Z}_n$  zusammen mit der Relation  $\text{mod}+$  eine Gruppe, denn es gelten die folgenden Eigenschaften einer Gruppe für alle Elemente von  $\mathbb{Z}_n$ :

- $a \text{ mod}+ b$  ist ein Element von  $\mathbb{Z}_n$  (Abgeschlossenheit),
- $(a \text{ mod}+ b) \text{ mod}+ c \equiv a \text{ mod}+ (b \text{ mod}+ c) \pmod{n}$  ( $\text{mod}+$  ist assoziativ),
- das neutrale Element ist die 0.
- jedes Element  $a \in \mathbb{Z}_n$  besitzt bezüglich dieser Operation ein Inverses, nämlich  $n - a$  (denn es gilt:  $a \text{ mod}+ (n - a) \equiv a + (n - a) \pmod{n} \equiv n \equiv 0 \pmod{n}$ ).

Da die Operation kommutativ ist, d.h. es gilt  $(a \text{ mod}+ b) = (b \text{ mod}+ a)$ , ist diese Struktur sogar eine „kommutative Gruppe“.

## 4.7.2 Multiplikation in einer Gruppe

Definiert man in der Menge  $\mathbb{Z}_n$  die Operation  $\text{mod}^*$  mit

$$a \text{ mod}^* b := (a * b) \pmod{n},$$

so ist  $\mathbb{Z}_n$  zusammen mit dieser Operation **normalerweise keine** Gruppe, weil nicht für jedes  $n$  alle Eigenschaften erfüllt sind.

**Beispiel:**

- a) In  $\mathbb{Z}_{15}$  besitzt z.B. das Element 5 kein Inverses. Es gibt nämlich kein  $a$  mit  $5 * a \equiv 1 \pmod{15}$ . Jedes Modulo-Produkt mit 5 ergibt auf dieser Menge 5, 10 oder 0.
- b) In  $\mathbb{Z}_{55} \setminus \{0\}$  besitzen z.B. die Elemente 5 und 11 keine multiplikativen Inversen. Es gibt nämlich kein  $a$  aus  $\mathbb{Z}_{55}$  mit  $5 * a \equiv 1 \pmod{55}$  und kein  $a$  mit  $11 * a \equiv 1 \pmod{55}$ . Das liegt daran, dass 5 und 11 nicht teilerfremd zu 55 sind. Jedes Modulo-Produkt mit 5 ergibt auf dieser Menge 5, 10, 15,  $\dots$ , 50 oder 0. Jedes Modulo-Produkt mit 11 ergibt auf dieser Menge 11, 22, 33, 44 oder 0.

Dagegen gibt es Teilmengen von  $\mathbb{Z}_n$ , die bezüglich  $\text{mod}^*$  eine Gruppe bilden. Wählt man sämtliche Elemente aus  $\mathbb{Z}_n$  aus, die teilerfremd zu  $n$  sind, so ist diese Menge eine Gruppe bezüglich  $\text{mod}^*$ . Diese Menge bezeichnet man mit  $\mathbb{Z}_n^*$ .

**Definition 4.7.2.**  $\mathbb{Z}_n^*$  :

$$\mathbb{Z}_n^* := \{a \in \mathbb{Z}_n \mid \text{ggT}(a, n) = 1\}.$$

$\mathbb{Z}_n^*$  wird manchmal auch als *reduzierte Restmenge*  $R'$  modulo  $n$  bezeichnet.

**Beispiel:** Für  $n = 10 = 2 * 5$  gilt:

vollständige Restmenge  $R = \mathbb{Z}_n = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ .

reduzierte Restmenge  $R' = \mathbb{Z}_n^* = \{1, 3, 7, 9\} \longrightarrow J(n) = 4$ .

**Bemerkung:**

$R'$  bzw.  $\mathbb{Z}_n^*$  ist immer eine echte Teilmenge von  $R$  bzw.  $\mathbb{Z}_n$ , da 0 immer Element von  $\mathbb{Z}_n$ , aber nie Element von  $\mathbb{Z}_n^*$  ist. Da 1 (per Definition) und  $n - 1$  immer teilerfremd zu  $n$  sind, sind sie stets Elemente beider Mengen.

Wählt man irgendein Element aus  $\mathbb{Z}_n^*$  und multipliziert es mit jedem anderen Element von  $\mathbb{Z}_n^*$ , so sind die Produkte<sup>27</sup> alle wieder in  $\mathbb{Z}_n^*$  und außerdem sind die Ergebnisse eine eindeutige Permutation der Elemente von  $\mathbb{Z}_n^*$ . Da die 1 immer Element von  $\mathbb{Z}_n^*$  ist, gibt es in dieser Menge eindeutig einen „Partner“, so dass das Produkt 1 ergibt. Mit anderen Worten:

**Satz 4.7.1.** *Jedes Element in  $\mathbb{Z}_n^*$  hat eine multiplikative Inverse.*

**Beispiel:**

$a = 3$  modulo 10 mit  $\mathbb{Z}_n^* = \{1, 3, 7, 9\}$  gilt  $a^{-1} = 7$ :

$$3 \equiv 3 * 1 \pmod{10},$$

$$9 \equiv 3 * 3 \pmod{10},$$

$$1 \equiv 3 * 7 \pmod{10},$$

$$7 \equiv 3 * 9 \pmod{10}.$$

<sup>27</sup> Dies ergibt sich aus der Abgeschlossenheit von  $\mathbb{Z}_n^*$  bezüglich der Multiplikation und der ggT-Eigenschaft:

$[a, b \in \mathbb{Z}_n^*] \Rightarrow [(a * b) \pmod{n}] \in \mathbb{Z}_n^*$ , genauer:

$[a, b \in \mathbb{Z}_n^*] \Rightarrow [\text{ggT}(a, n) = 1, \text{ggT}(b, n) = 1] \Rightarrow [\text{ggT}(a * b, n) = 1] \Rightarrow [(a * b) \pmod{n}] \in \mathbb{Z}_n^*$ .

Die eindeutige Invertierung (Umkehrbarkeit) ist eine notwendige Bedingung für die Kryptographie (siehe Kapitel 4.10: Beweis des RSA-Verfahrens mit Euler-Fermat).



Eric Berne<sup>28</sup>:

Die mathematische Spielanalyse postuliert Spieler, die rational reagieren. Die transaktionale Spielanalyse dagegen befasst sich mit Spielen, die unrational, ja sogar **irrational und damit wirklichkeitsnäher** sind.

## 4.8 Euler-Funktion, kleiner Satz von Fermat und Satz von Euler-Fermat

### 4.8.1 Muster und Strukturen

So wie Mathematiker die Struktur  $a * x \equiv b \pmod{m}$  untersuchen (s. Kapitel 4.5.2), so interessiert sie auch die Struktur  $x^a \equiv b \pmod{m}$ .

Auch hierbei ist insbesondere der Fall interessant, wenn  $b = 1$  ist (also den Wert der multiplikativen Einheit annimmt) und wenn  $b = x$  ist (also die Abbildung einen Fixpunkt hat).

### 4.8.2 Die Euler-Funktion

Bei vorgegebenem  $n$  ist die Anzahl der Zahlen aus der Menge  $\{1, \dots, n-1\}$ , die zu  $n$  teilerfremd sind, gleich dem Wert der Euler<sup>29</sup>-Funktion  $J(n)$ .

**Definition 4.8.1.** Die Euler-Funktion<sup>30</sup>  $J(n)$  gibt die Anzahl der Elemente von  $\mathbb{Z}_n^*$  an.

$J(n)$  gibt auch an, wieviele ganze Zahlen in  $\pmod{n}$  multiplikative Inverse haben.  $J(n)$  lässt sich berechnen, wenn man die Primfaktorzerlegung von  $n$  kennt.

**Satz 4.8.1.** Für eine Primzahl gilt:  $J(p) = p - 1$ .

**Satz 4.8.2.** Ist  $n$  das Produkt zweier verschiedenen Primzahlen  $p$  und  $q$ , so gilt:

$$J(p * q) = (p - 1) * (q - 1) \quad \text{oder} \quad J(p * q) = J(p) * J(q).$$

Dieser Fall ist für das RSA-Verfahren wichtig.

**Satz 4.8.3.** Ist  $n = p_1 * p_2 * \dots * p_k$ , wobei  $p_1$  bis  $p_k$  verschiedene Primzahlen sind (d.h.  $p_i \neq p_j$  für  $i \neq j$ ), dann gilt (als Verallgemeinerung von Satz 4.8.2):

$$J(n) = (p_1 - 1) * (p_2 - 1) * \dots * (p_k - 1).$$

**Satz 4.8.4.** Verallgemeinert gilt für jede Primzahl  $p$  und jedes  $n$  aus  $\mathbb{N}$ :

1.  $J(p^n) = p^{n-1} * (p - 1)$ .

2. Ist  $n = p_1^{e_1} * p_2^{e_2} * \dots * p_k^{e_k}$ , wobei  $p_1$  bis  $p_k$  verschiedene Primzahlen sind, dann gilt:

$$J(n) = [(p_1^{e_1-1}) * (p_1 - 1)] * \dots * [(p_k^{e_k-1}) * (p_k - 1)] = n * [(p_1 - 1)/p_1] * \dots * [(p_k - 1)/p_k].$$

<sup>28</sup>Eric Berne, „Spiele der Erwachsenen“, rororo, (c) 1964, S. 235.

<sup>29</sup>Leonhard Euler, schweizer Mathematiker, 15.4.1707 – 18.9.1783

<sup>30</sup>Wird oft auch als Eulersche Phi-Funktion  $\Phi(n)$  geschrieben.

### Beispiel:

- $n = 70 = 2 * 5 * 7 \implies$  nach Satz 4.8.3:  $J(n) = 1 \cdot 4 \cdot 6 = 24$ .
- $n = 9 = 3^2 \implies$  nach Satz 4.8.4:  $J(n) = 3^1 \cdot 2 = 6$ , weil  $\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$ .
- $n = 2.701.125 = 3^2 * 5^3 * 7^4 \implies$  nach Satz 4.8.4:  $J(n) = [3^1 * 2] * [5^2 * 4] * [7^3 * 6] = 1.234.800$ .

### 4.8.3 Der Satz von Euler-Fermat

Für den Beweis des RSA-Verfahrens brauchen wir den Satz von Fermat und dessen Verallgemeinerung (Satz von Euler-Fermat) – vergleiche Kapitel 3.5.

**Satz 4.8.5. Kleiner Satz von Fermat**<sup>31</sup> Sei  $p$  eine Primzahl und  $a$  eine beliebige ganze Zahl, dann gilt

$$a^p \equiv a \pmod{p}.$$

Eine alternative Formulierung des kleinen Satzes von Fermat lautet: Sei  $p$  eine Primzahl und  $a$  eine beliebige ganze Zahl, die teilerfremd zu  $p$  ist, dann gilt:

$$a^{p-1} \equiv 1 \pmod{p}.$$

**Satz 4.8.6. Satz von Euler-Fermat (Verallgemeinerung des kleinen Satzes von Fermat)** Für alle Elemente  $a$  aus der Gruppe  $\mathbb{Z}_n^*$  gilt (d.h.  $a$  und  $n$  sind natürliche Zahlen, die teilerfremd zueinander sind):

$$a^{J(n)} \equiv 1 \pmod{n}.$$

Dieser Satz besagt, dass wenn man ein Gruppenelement (hier  $a$ ) mit der Ordnung der Gruppe (hier  $J(n)$ ) potenziert, ergibt sich immer das neutrale Element der Multiplikation (die Zahl 1).

Die 2. Formulierung des kleinen Satzes von Fermat ergibt sich direkt aus dem Satz von Euler, wenn  $n$  eine Primzahl ist.

Falls  $n$  das Produkt zweier verschiedenen Primzahlen ist, kann man mit dem Satz von Euler in bestimmten Fällen sehr schnell das Ergebnis einer modularen Potenz berechnen. Es gilt:  $a^{(p-1)*(q-1)} \equiv 1 \pmod{pq}$ .

### Beispiel Berechnung einer modularen Potenz:

- Mit  $2 = 1 * 2$  und  $6 = 2 * 3$ , wobei 2 und 3 jeweils prim;  $J(6) = 2$ , da nur 1 und 5 zu 6 teilerfremd sind, folgt  $5^2 \equiv 5^{J(6)} \equiv 1 \pmod{6}$ , ohne dass man die Potenz berechnen musste.
- Mit  $792 = 22 * 36$  und  $23 * 37 = 851$ , wobei 23 und 37 jeweils prim sind, folgt für  $31 \in \mathbb{Z}_{851}^*$   $31^{792} \equiv 31^{J(23*37)} \equiv 31^{J(851)} \equiv 1 \pmod{851}$ .

### 4.8.4 Bestimmung der multiplikativen Inversen

Eine weitere interessante Anwendung ist ein Sonderfall der Bestimmung der multiplikativen Inverse mit Hilfe des Satzes von Euler-Fermat (multiplikative Inverse werden ansonsten mit dem erweiterten Euklid'schen Algorithmus ermittelt).

---

<sup>31</sup>Pierre de Fermat, französischer Mathematiker, 17.8.1601 – 12.1.1665.

**Beispiel:**

Finde die multiplikative Inverse von 1579 modulo 7351.

Nach Euler-Fermat gilt:  $a^{J(n)} = 1 \pmod n$  für alle  $a$  aus  $\mathbb{Z}_n^*$ . Teilt man beide Seiten durch  $a$ , ergibt sich:  $a^{J(n)-1} \equiv a^{-1} \pmod n$ . Für den Spezialfall, dass der Modul prim ist, gilt  $J(n) = p - 1$ . Also gilt für die modulare Inverse  $a^{-1}$  von  $a$ :

$$a^{-1} \equiv a^{J(n)-1} \equiv a^{(p-1)-1} \equiv a^{p-2} \pmod p.$$

Für unser Beispiel bedeutet das:

Da der Modul 7351 prim ist, ist  $p - 2 = 7349$ .

$$1579^{-1} \equiv 1579^{7349} \pmod p.$$

Durch geschicktes Zerlegen des Exponenten kann man diese Potenz relativ einfach berechnen (siehe Kapitel 4.6.4 Schnelles Berechnen hoher Potenzen):

$$\begin{aligned} 7349 &= 4096 + 2048 + 1024 + 128 + 32 + 16 + 4 + 1 \\ 1579^{-1} &\equiv 1579^{7349} \pmod{7351}. \end{aligned}$$

**4.8.5 Fixpunkte modulo 26**

Laut Satz 4.6.3 werden die arithmetischen Operationen von modularen Ausdrücken in den Exponenten modulo  $J(n)$  und nicht modulo  $n$  durchgeführt<sup>32</sup>.

Wenn man in  $a^{e*d} \equiv a^1 \pmod n$  die Inverse z.B. für den Faktor  $e$  im Exponenten bestimmen will, muss man modulo  $J(n)$  rechnen.

**Beispiel:** (mit Bezug zum RSA-Algorithmus)

Wenn man modulo 26 rechnet, aus welcher Menge können  $e$  und  $d$  kommen?

Lösung: Es gilt  $e * d \equiv 1 \pmod{J(26)}$ .

Die reduzierte Restmenge  $R' = \mathbb{Z}_{26}^* = \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$  sind die Elemente in  $\mathbb{Z}_{26}$ , die eine multiplikative Inverse haben, also teilerfremd zu 26 sind.

Die reduzierte Restmenge  $R''$  enthält nur die Elemente aus  $R'$ , die teilerfremd zu  $J(26) = 12$  sind:  $R'' = \{1, 5, 7, 11\}$ .

Für jedes  $e$  aus  $R''$  gibt es ein  $d$  aus  $R''$ , so dass  $a \equiv (a^e)^d \pmod n$ .

Somit gibt es also zu jedem  $e$  in  $R''$  genau ein (nicht unbedingt von  $e$  verschiedenes) Element, so dass gilt:  $e * d \equiv 1 \pmod{J(26)}$ .

Für alle  $e$ , die teilerfremd zu  $J(n)$  sind, könnte man nach dem Satz von Euler-Fermat das  $d$  folgendermaßen berechnen:

$$\begin{aligned} d &\equiv e^{-1} \pmod{J(n)} \\ &\equiv e^{J(n)-1} \pmod{J(n)}, \quad \text{denn } a^{J(n)} \equiv 1 \pmod n \quad \text{entspricht } a^{J(n)-1} \equiv a^{-1} \pmod n. \end{aligned}$$

Besteht die Zahl  $n$  aus zwei verschiedenen Primfaktoren, so ist die Faktorisierung von  $n$  und das Finden von  $J(n)$  ähnlich schwierig<sup>33</sup> (vergleiche Forderung 3 in Kapitel 4.10.1).

<sup>32</sup>Für das folgende Beispiel wird der Modul wie beim RSA-Verfahren üblich mit „ $n$ “ statt mit „ $m$ “ bezeichnet.

<sup>33</sup>Für  $n = pq$  mit  $p \neq q$  gilt  $J(n) = (p - 1) * (q - 1) = n - (p + q) + 1$ . Ferner sind die Zahlen  $p$  und  $q$  Lösungen

## 4.9 Multiplikative Ordnung und Primitivwurzel<sup>34</sup>

Die Multiplikative Ordnung (order) und die Primitivwurzel (primitive root) sind zwei nützliche Konstrukte (Konzepte) der elementaren Zahlentheorie.

Mathematiker stellen sich die Frage, unter welchen Bedingungen ergibt die wiederholte Anwendung einer Operation das neutrale Element (vergleiche Muster und Strukturen, Kapitel 4.8.1).

Für die  $i$ -fach aufeinander folgende modulare Multiplikation einer Zahl  $a$  für  $i = 1, \dots, m - 1$  ergibt sich als Produkt das neutrale Element der Multiplikation nur dann, wenn  $a$  und  $m$  teilerfremd sind.

**Definition 4.9.1.** Die **multiplikative Ordnung**  $\text{ord}_m(a)$  einer ganzen Zahl  $a \pmod{m}$  (wobei  $a$  und  $m$  teilerfremd sind) ist die kleinste ganze Zahl  $i$ , für die gilt:  $a^i \equiv 1 \pmod{m}$ .

Die folgende Tabelle 4.7 zeigt, dass in einer multiplikativen Gruppe (hier  $\mathbb{Z}_{11}^*$ ) nicht notwendig alle Zahlen die gleiche Ordnung haben: Die Ordnungen sind 1, 2, 5 und 10. Dabei bemerkt man:

1. Die Ordnungen sind alle Teiler von 10.
2. Die Zahlen  $a = 2, 6, 7$  und 8 haben die Ordnung 10. Man sagt diese Zahlen haben in  $\mathbb{Z}_{11}^*$  **maximale Ordnung**.

### Beispiel 1:

Die folgende Tabelle 4.7<sup>35</sup> zeigt die Werte  $a^i \pmod{11}$  für die Exponenten  $i = 1, 2, \dots, 10$  und für die Basen  $a = 1, 2, \dots, 10$  sowie den sich für jedes  $a$  daraus ergebenden Wert  $\text{ord}_{11}(a)$ :

Aus der Tabelle 4.7 kann man ersehen, dass z.B. die Ordnung von 3 modulo 11 den Wert 5 hat.

**Definition 4.9.2.** Sind  $a$  und  $m$  teilerfremd und gilt  $\text{ord}_m(a) = J(m)$ , (d.h.  $a$  hat maximale Ordnung), dann nennt man  $a$  eine **Primitivwurzel** von  $m$ .

Nicht zu jedem Modul  $m$  gibt es eine Zahl  $a$ , die eine Primitivwurzel ist. In der obigen Tabelle 4.7 ist nur  $a = 2, 6, 7$  und 8 bezüglich mod 11 eine Primitivwurzel ( $J(11) = 10$ ).

Mit Hilfe der Primitivwurzel kann man die Bedingungen klar herausarbeiten, wann Potenzen modulo  $m$  *eindeutig invertierbar* und die Berechnung in den Exponenten handhabbar sind.

Die folgenden beiden Tabellen zeigen multiplikative Ordnung und Primitivwurzel modulo 45 und modulo 46.

---

der quadratischen Gleichung  $x^2 - (p + q)x + pq = 0$ .

Sind nur  $n$  und  $J(n)$  bekannt, so gilt  $pq = n$  und  $p + q = n + 1 - J(n)$ . Man erhält somit die Faktoren  $p$  und  $q$  von  $n$ , indem man die folgende quadratische Gleichung löst:

$$x^2 + (J(n) - n - 1)x + n = 0$$

<sup>34</sup>Mit dem Lernprogramm **ZT** können Sie einige der hier besprochenen Verfahren vertiefen (siehe Lern-Kapitel 2.2, Seiten 10-14/40 und 24-40/40).

ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

<sup>35</sup>In Kapitel 4.18.3 “Multiplikative Ordnung” finden Sie den Quelltext zur Bestimmung der Tabelle mit Sage.

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	$ord_{11}(a)$
a=1	1	1	1	1	1	1	1	1	1	1	1
a=2	2	4	8	5	10	9	7	3	6	1	10
a=3	3	9	5	4	1	3	9	5	4	1	5
a=4	4	5	9	3	1	4	5	9	3	1	5
a=5	5	3	4	9	1	5	3	4	9	1	5
a=6	6	3	7	9	10	5	8	4	2	1	10
a=7	7	5	2	3	10	4	6	9	8	1	10
a=8	8	9	6	4	10	3	2	5	7	1	10
a=9	9	4	3	5	1	9	4	3	5	1	5
a=10	10	1	10	1	10	1	10	1	10	1	2

Tabelle 4.7: Werte von  $a^i \bmod 11$ ,  $1 \leq a, i < 11$  und zugehörige Ordnung von  $a$  modulo  $m$

### Beispiel 2:

Die folgende Tabelle 4.8<sup>36</sup> zeigt die Werte  $a^i \bmod 45$  für die Exponenten  $i = 1, 2, \dots, 12$  und für die Basen  $a = 1, 2, \dots, 12$  sowie den sich für jedes  $a$  daraus ergebenden Wert  $ord_{45}(a)$ .

$a \setminus i$	1	2	3	4	5	6	7	8	9	10	11	12	$ord_{45}(a)$	$J(45)$
1	1	1	1	1	1	1	1	1	1	1	1	1	1	24
2	2	4	8	16	32	19	38	31	17	34	23	1	12	24
3	3	9	27	36	18	9	27	36	18	9	27	36	—	24
4	4	16	19	31	34	1	4	16	19	31	34	1	6	24
5	5	25	35	40	20	10	5	25	35	40	20	10	—	24
6	6	36	36	36	36	36	36	36	36	36	36	36	—	24
7	7	4	28	16	22	19	43	31	37	34	13	1	12	24
8	8	19	17	1	8	19	17	1	8	19	17	1	4	24
9	9	36	9	36	9	36	9	36	9	36	9	36	—	24
10	10	10	10	10	10	10	10	10	10	10	10	10	—	24
11	11	31	26	16	41	1	11	31	26	16	41	1	6	24
12	12	9	18	36	27	9	18	36	27	9	18	36	—	24

Tabelle 4.8: Werte von  $a^i \bmod 45$ ,  $1 \leq a, i < 13$

$J(45)$  berechnet sich nach Satz 4.8.4:  $J(45) = J(3^2 * 5) = [3^1 * 2] * [1 * 4] = 24$ .

Da 45 keine Primzahl ist, gibt es nicht für alle Werte von  $a$  eine "Multiplikative Ordnung" (für alle nicht zu 45 teilerfremden Zahlen 3, 5, 6, 9, 10, 12,  $\dots$ , da  $45 = 3^2 * 5$ ).

### Beispiel 3:

Hat 7 eine Primitivwurzel modulo 45?

Die notwendige, aber nicht hinreichende Voraussetzung/Bedingung  $\text{ggT}(7, 45) = 1$  ist erfüllt. Aus der Tabelle 4.8 kann man ersehen, dass die Zahl  $a = 7$  keine Primitivwurzel von 45 ist, denn  $ord_{45}(7) = 12 \neq 24 = J(45)$ .

<sup>36</sup>In Kapitel 4.18.3 "Multiplikative Ordnung" finden Sie den Quelltext zur Berechnung der Tabelle mit Sage.

#### Beispiel 4:

Die folgende Tabelle 4.9<sup>37</sup> beantwortet die Frage, ob die Zahl  $a = 7$  eine Primitivwurzel von 46 ist. Die notwendige, aber nicht hinreichende Voraussetzung/Bedingung  $\text{ggT}(7, 46) = 1$  ist erfüllt.

$J(46)$  berechnet sich nach Satz 4.8.2:  $J(46) = J(2 * 23) = 1 * 22 = 22$ . Die Zahl 7 ist eine Primitivwurzel von 46, denn  $\text{ord}_{46}(7) = 22 = J(46)$ .

$a \setminus i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	ord
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	32	18	36	26	6	12	24	2	4	8	16	32	18	36	26	6	12	24	2	–
3	3	9	27	35	13	39	25	29	41	31	1	3	9	27	35	13	39	25	29	41	31	1	3	11
4	4	16	18	26	12	2	8	32	36	6	24	4	16	18	26	12	2	8	32	36	6	24	4	–
5	5	25	33	27	43	31	17	39	11	9	45	41	21	13	19	3	15	29	7	35	37	1	5	22
6	6	36	32	8	2	12	26	18	16	4	24	6	36	32	8	2	12	26	18	16	4	24	6	–
7	7	3	21	9	17	27	5	35	15	13	45	39	43	25	37	29	19	41	11	31	33	1	7	22
8	8	18	6	2	16	36	12	4	32	26	24	8	18	6	2	16	36	12	4	32	26	24	8	–
9	9	35	39	29	31	3	27	13	25	41	1	9	35	39	29	31	3	27	13	25	41	1	9	11
10	10	8	34	18	42	6	14	2	20	16	22	36	38	12	28	4	40	32	44	26	30	24	10	–
11	11	29	43	13	5	9	7	31	19	25	45	35	17	3	33	41	37	39	15	27	21	1	11	22
12	12	6	26	36	18	32	16	8	4	2	24	12	6	26	36	18	32	16	8	4	2	24	12	–
13	13	31	35	41	27	29	9	25	3	39	1	13	31	35	41	27	29	9	25	3	39	1	13	11
14	14	12	30	6	38	26	42	36	44	18	22	32	34	16	40	8	20	4	10	2	28	24	14	–
15	15	41	17	25	7	13	11	27	37	3	45	31	5	29	21	39	33	35	19	9	43	1	15	22
16	16	26	2	32	6	4	18	12	8	36	24	16	26	2	32	6	4	18	12	8	36	24	16	–
17	17	13	37	31	21	35	43	41	7	27	45	29	33	9	15	25	11	3	5	39	19	1	17	22
18	18	2	36	4	26	8	6	16	12	32	24	18	2	36	4	26	8	6	16	12	32	24	18	–
19	19	39	5	3	11	25	15	9	33	29	45	27	7	41	43	35	21	31	37	13	17	1	19	22
20	20	32	42	12	10	16	44	6	28	8	22	26	14	4	34	36	30	2	40	18	38	24	20	–
21	21	27	15	39	37	41	33	3	17	35	45	25	19	31	7	9	5	13	43	29	11	1	21	22
22	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	–
23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	–

Tabelle 4.9: Werte von  $a^i \bmod 46, 1 \leq a, i < 23$

**Satz 4.9.1.** <sup>38,39</sup> Für einen Modul  $n$  und eine Zahl  $a$ , teilerfremd zu  $n$  gilt:

Die Menge  $\{a^i \bmod n \mid i = 1, \dots, J(n)\}$  ist gleich der multiplikativen Gruppe  $Z_n^*$  genau dann, wenn  $\text{ord}_n(a) = J(n)$ .

Die multiplikative Gruppe  $Z_n^*$  nimmt alle Werte von 1 bis  $n - 1$  an.

<sup>37</sup>In Kapitel 4.18.3 “Multiplikative Ordnung” finden Sie den Quelltext zur Berechnung der Tabelle mit Sage.

<sup>38</sup>Für Primmoduli  $p$  haben alle  $a$  mit  $0 < a < p$  die Ordnung  $J(p) = p - 1$ . Vergleiche dazu Tabelle 4.8. In diesem Fall nimmt  $a^i \bmod p$  alle Werte  $1, \dots, p - 1$  an. Dieses Ausschöpfen des Wertebereiches ist eine wichtige kryptographische Eigenschaft (vergleiche Satz 4.6.2). Hiermit wird eine Permutation  $\pi(p - 1)$  festgelegt.

<sup>39</sup>Tabelle 4.9 zeigt, dass bei zusammengesetzten Moduli  $n$  nicht alle  $a$  die maximale Ordnung  $J(n)$  haben. In diesem Beispiel haben nur 5, 7, 11, 15, 17, 19 und 21 die Ordnung 22.

## 4.10 Beweis des RSA-Verfahrens mit Euler-Fermat

Mit dem Satz von Euler-Fermat kann man in der Gruppe  $\mathbb{Z}_n^*$  das RSA<sup>40,41</sup>-Verfahren „beweisen“.

### 4.10.1 Grundidee der Public-Key-Kryptographie

Die Grundidee bei der Public-Key-Kryptographie besteht darin, dass alle Teilnehmer ein unterschiedliches Paar von Schlüsseln ( $P$  und  $S$ ) besitzen und man für alle Empfänger die öffentlichen Schlüssel publiziert. So wie man die Telefonnummer einer Person aus dem Telefonbuch nachschlägt, kann man den öffentlichen Schlüssel  $P$  (public) des Empfängers aus einem Verzeichnis entnehmen. Außerdem hat jeder Empfänger einen geheimen Schlüssel  $S$  (secret), der zum Entschlüsseln benötigt wird und den niemand sonst kennt. Möchte der Sender eine Nachricht  $M$  (message) schicken, verschlüsselt er diese Nachricht mit dem öffentlichen Schlüssel  $P$  des Empfängers, bevor er sie abschickt:

Der Chiffretext  $C$  (ciphertext) ergibt sich mit  $C = E(P; M)$ , wobei  $E$  (encryption) die Verschlüsselungsvorschrift ist. Der Empfänger benutzt seinen privaten Schlüssel  $S$ , um die Nachricht wieder mit der Entschlüsselungsvorschrift  $D$  (decryption) zu entschlüsseln:  $M = D(S; C)$ .

Damit dieses System mit jeder Nachricht  $M$  funktioniert, müssen folgende 4 **Forderungen** erfüllt sein:

1.  $D(S; E(P; M)) = M$  für jedes  $M$  (Umkehrbarkeit) und  $M$  nimmt „sehr viele“ verschiedene Werte an.
2. Alle  $(S, P)$ -Paare aller Teilnehmer sind verschieden (d.h. es muss viele davon geben).
3. Der Aufwand,  $S$  aus  $P$  herzuleiten, ist mindestens so hoch, wie das Entschlüsseln von  $M$  ohne Kenntnis von  $S$ .
4. Sowohl  $C$  als auch  $M$  lassen sich relativ einfach berechnen.

Die 1. Forderung ist eine generelle Bedingung für alle kryptographischen Verschlüsselungsalgorithmen.

Die 2. Forderung kann leicht sichergestellt werden, weil es „sehr“ viele Primzahlen gibt<sup>42</sup> und weil dies durch eine zentrale Stelle, die Zertifikate ausgibt, sichergestellt werden kann.

Die letzte Forderung macht das Verfahren überhaupt erst anwendbar. Dies geht, weil die modulare Potenzierung in linearer Zeit möglich ist (da die Zahlen längenbeschränkt sind).

---

<sup>40</sup>Das RSA-Verfahren ist das verbreitetste asymmetrische Kryptoverfahren. Es wurde 1978 von Ronald Rivest, Adi Shamir und Leonard Adleman entwickelt und kann sowohl zum Signieren wie zum Verschlüsseln eingesetzt werden. Kryptographen assoziieren mit der Abkürzung „**RSA**“ immer dieses Verfahren – die folgende Anmerkung soll eher humorvoll zeigen, dass man jede Buchstabenkombination mehrfach belegen kann: In Deutschland gibt es sehr Interessen-lastige Diskussionen im Gesundheitswesen. Dabei wird mit „RSA“ der vom Gesundheitsministerium geregelte „**RisikoStrukturAusgleich**“ in der gesetzlichen Krankenversicherung: Im Jahr 2004 wurden durch den RSA ca. 16 Mrd. Euro zwischen den Krankenkassen umverteilt.

<sup>41</sup>In Literatur und in Filmen sind sowohl klassische als auch moderne Kryptographie-Verfahren verarbeitet worden (siehe Anhang A.3).

<sup>42</sup>Nach dem **Primzahlsatz** (Kapitel 3.7.2, S. 70) von Legendre und Gauss gibt es bis zur Zahl  $n$  asymptotisch  $n/\ln(n)$  Primzahlen. Dies bedeutet beispielsweise: es gibt  $6,5 \cdot 10^{74}$  Primzahlen unterhalb von  $n = 2^{256}$  ( $1,1 \cdot 10^{77}$ ) und  $3,2 \cdot 10^{74}$  Primzahlen unterhalb von  $n = 2^{255}$ . Zwischen  $2^{256}$  und  $2^{255}$  gibt es also allein  $3,3 \cdot 10^{74}$  Primzahlen mit genau 256 Bits. Diese hohe Zahl ist auch der Grund, warum man sie nicht einfach alle abspeichern kann.

Während Whitfield Diffie und Martin Hellman schon 1976 das generelle Schema formulierten, fanden erst Rivest, Shamir und Adleman ein konkretes Verfahren, das alle vier Bedingungen erfüllte.

#### 4.10.2 Funktionsweise des RSA-Verfahrens

Man kann die Einzelschritte zur Durchführung des RSA-Verfahrens folgendermaßen beschreiben (s. [Eckert2003, S. 213 ff] und [Sedgewick1990, S. 338 ff]). Schritt 1 bis 3 sind die Schlüsselerzeugung, Schritt 4 und 5 sind die Verschlüsselung, 6 und 7 die Entschlüsselung:

1. Wähle zufällig 2 verschiedene Primzahlen<sup>43,44</sup>  $p$  und  $q$  und berechne  $n = p * q$ <sup>45</sup>.  
Der Wert  $n$  wird als RSA-Modul bezeichnet<sup>46</sup>.
2. Wähle ein beliebiges  $e \in \{2, \dots, n-1\}$ , so dass gilt<sup>47</sup>:  
 $e$  ist teilerfremd zu  $J(n) = (p-1) * (q-1)$ .  
Danach kann man  $p$  und  $q$  „wegwerfen“.<sup>48</sup>
3. Wähle  $d \in \{1, \dots, n-1\}$  mit  $e * d = 1 \bmod J(n)$ , d.h.  $d$  ist die multiplikative Inverse zu  $e$  modulo  $J(n)$ <sup>49,50</sup>. Danach kann man  $J(n)$  „wegwerfen“.  
 $\rightarrow (n, e)$  ist der öffentliche Schlüssel  $P$ .  
 $\rightarrow (n, d)$  ist der geheime Schlüssel  $S$  (es ist nur  $d$  geheim zu halten).
4. Zum Verschlüsseln wird die als (binäre) Zahl dargestellte Nachricht in Teile aufgebrochen, so dass jede Teilzahl kleiner als  $n$  ist.
5. Verschlüsselung des Klartextes (bzw. seiner Teilstücke)  $M \in \{1, \dots, n-1\}$ :

$$C = E((n, e); M) := M^e \bmod n.$$

<sup>43</sup>Compaq hatte in 2000 mit hohem Marketingaufwand das sogenannte Multiprime-Verfahren eingeführt.  $n$  war dabei das Produkt von zwei großen und einer relativ dazu kleinen Primzahl:  $n = o * p * q$ . Nach Satz 4.8.3 ist dann  $J(n) = (o-1) * (p-1) * (q-1)$ . Das Verfahren hat sich bisher nicht durchgesetzt.

Mit ein Grund dafür dürfte sein, dass Compaq ein Patent dafür angemeldet hat. Generell gibt es in Europa und in der Open Source Bewegung wenig Verständnis für Patente auf Algorithmen. Überhaupt kein Verständnis herrscht außerhalb der USA, dass man auf den Sonderfall (3 Faktoren) eines Algorithmus (RSA) ein Patent beantragen kann, obwohl das Patent für den allgemeinen Fall schon fast abgelaufen war.

<sup>44</sup>Für Primzahlen  $p$  und  $q$  mit  $p = q$ , und  $e, d$  mit  $ed \equiv 1 \bmod J(n)$  gilt i.a. nicht  $(m^e)^d \equiv m \bmod n$  für alle  $m < n$ . Sei z.B.  $n = 5^2$ , berechnet sich  $J(n)$  nach Satz 4.8.4:  $J(n) = 5 * 4 = 20$ ,  $e = 3$ ,  $d = 7$ ,  $ed \equiv 21 \equiv 1 \bmod J(n)$ . Dann gilt  $(5^3)^7 \equiv 0 \bmod 25$ .

<sup>45</sup>Das BSI (Bundesamt für Sicherheit in der Informationstechnik) empfiehlt, die Primfaktoren  $p$  und  $q$  ungefähr gleich groß zu wählen, aber nicht zu dicht beieinander, d.h. konkret etwa

$$0.5 < |\log_2(p) - \log_2(q)| < 30.$$

Die Primfaktoren werden unter Beachtung der genannten Nebenbedingung zufällig und unabhängig voneinander erzeugt (Siehe <http://www.bsi.bund.de/esig/basics/techbas/krypto/bund02v7.pdf>).

<sup>46</sup>In CrypTool wird der RSA-Modul mit einem großen „N“ bezeichnet.

<sup>47</sup>Empfehlenswert aus kryptoanalytischen Gründen, aber nicht notwendig für das Funktionieren des Verfahrens ist,  $e$  so zu wählen, dass gilt:  $\max(p, q) < e < J(n) - 1$ .

<sup>48</sup>Das Verfahren erlaubt es auch,  $d$  frei zu wählen und dann  $e$  zu berechnen. Dies hat aber praktische Nachteile. Normalerweise will man „schnell“ verschlüsseln können und wählt deshalb einen öffentlichen Exponenten  $e$  so, dass er im Vergleich zum Modul  $n$  sehr kleine Bitlängen und möglichst wenige binäre Einsen hat (z.B.  $2^{16} + 1$ ). Damit ist eine schnelle Exponentiation bei der Verschlüsselung möglich. Als besonders praktisch haben sich hierfür die Primzahlen 3, 17 und 65537 erwiesen. Am häufigsten verwendet wird die Zahl  $65537 = 2^{16} + 1$ , also binär:  $10 \dots 0 \dots 01$  (diese Zahl ist prim und deshalb zu sehr vielen Zahlen teilerfremd).

<sup>49</sup>Aus Sicherheitsgründen darf  $d$  nicht zu klein sein.

<sup>50</sup>Je nach Implementierung wird zuerst  $d$  oder zuerst  $e$  bestimmt.



6. Zum Entschlüsseln wird das binär als Zahl dargestellte Chiffre in Teile aufgebrochen, so dass jede Teilzahl kleiner als  $n$  ist.
7. Entschlüsselung des Chiffretextes (bzw. seiner Teilstücke)  $C \in \{1, \dots, n-1\}$ :

$$M = D((n, d); C) := C^d \bmod n.$$

Die Zahlen  $d, e, n$  sind normalerweise sehr groß (z.B. sind  $d$  und  $e$  300 bit, und  $n$  600 bit lang).

**Bemerkung:**

Die Sicherheit des RSA-Verfahrens hängt wie bei allen Public-Key-Verfahren davon ab, dass man den privaten Key  $d$  nicht aus dem öffentlichen Key  $(n, e)$  berechnen kann.

Beim RSA-Verfahren bedeutet dies,

1. dass  $J(n)$  für große zusammengesetzte  $n$  schwer zu berechnen ist, und
2. dass  $n$  für große  $n$  nur schwer in seine Primfaktoren zerlegt werden kann (Faktorisierungsproblem).<sup>51</sup>

### 4.10.3 Beweis der Forderung 1 (Umkehrbarkeit)

Für Schlüsselpaare  $(n, e)$  und  $(n, d)$ , die die in den Schritten 1 bis 3 des RSA-Verfahrens festgelegten Eigenschaften besitzen, muss für alle  $M < n$  gelten:

$$M \equiv (M^e)^d \pmod{n} \quad \text{wobei} \quad (M^e)^d = M^{e*d}.$$

Das heißt, der oben angegebene Dechiffrieralgorithmus arbeitet korrekt.

Zu zeigen ist also:  $M^{e*d} = M \pmod{n}$ :

Wir zeigen das in 3 Schritten (s. [Beutelspacher1996, S. 131ff]).

**Schritt 1:**

Im ersten Schritt zeigen wir:

$$M^{e*d} \equiv M \pmod{p}.$$

Dies ergibt sich aus den Voraussetzungen und dem Satz von Euler-Fermat (Satz 4.8.6). Da  $n = p*q$  und  $J(p*q) = (p-1)*(q-1)$  und da  $e$  und  $d$  so gewählt sind, dass  $e*d \equiv 1 \pmod{J(n)}$ , gibt es eine ganze Zahl  $k$ , so dass gilt:  $e*d = 1 + k*(p-1)*(q-1)$ .

$$\begin{aligned} M^{e*d} &\equiv M^{1+k*J(n)} \equiv M * M^{k*J(n)} \equiv M * M^{k*(p-1)*(q-1)} \pmod{p} \\ &\equiv M * (M^{p-1})^{k*(q-1)} \pmod{p} \quad \text{aufgrund des kleinen Fermat: } M^{p-1} \equiv 1 \pmod{p} \\ &\equiv M * (1)^{k*(q-1)} \pmod{p} \\ &\equiv M \pmod{p}. \end{aligned}$$

Die Voraussetzung für die Anwendung des vereinfachten Satzes von Euler-Fermat (Kleiner-Fermat Satz 4.8.5) war, dass  $M$  und  $p$  teilerfremd sind.

<sup>51</sup>Für die manchmal geäußerte Sorge, dass es nicht genug Primzahlen gäbe, besteht kein Grund: Dass es immer ausreichend Primzahlen gibt, wenn man die Dimension (Exponenten) des Moduls hochsetzt, wird visualisiert in Kapitel 3.13 "Anhang: Visualisierung der Menge der Primzahlen in hohen Bereichen".

Da das im allgemeinen nicht gilt, müssen wir noch betrachten, was ist, wenn  $M$  und  $p$  nicht teilerfremd sind: Da  $p$  eine Primzahl ist, muss dann notwendigerweise  $p$  ein Teiler von  $M$  sein. Das heißt aber:

$$M \equiv 0 \pmod{p}$$

Wenn  $p$  die Zahl  $M$  teilt, so teilt  $p$  erst recht  $M^{e*d}$ . Also ist auch:

$$M^{e*d} \equiv 0 \pmod{p}.$$

Da  $p$  sowohl  $M$  als auch  $M^{e*d}$  teilt, teilt er auch ihre Differenz:

$$(M^{e*d} - M) \equiv 0 \pmod{p}.$$

Und damit gilt auch in diesem Spezialfall unsere zu beweisende Behauptung.

### Schritt 2:

Völlig analog beweist man:  $M^{e*d} \equiv M \pmod{q}$ .

### Schritt 3:

Nun führen wir die Behauptungen aus Schritt 1 und 2 zusammen für  $n = p*q$ , um zu zeigen:

$$M^{e*d} \equiv M \pmod{n} \text{ für alle } M < n.$$

Nach Schritt 1 und 2 gilt  $(M^{e*d} - M) \equiv 0 \pmod{p}$  und  $(M^{e*d} - M) \equiv 0 \pmod{q}$ , also teilen  $p$  und  $q$  jeweils dieselbe Zahl  $z = (M^{e*d} - M)$ . Da  $p$  und  $q$  **verschiedene** Primzahlen sind, muss dann auch ihr Produkt diese Zahl  $z$  teilen. Also gilt:

$$(M^{e*d} - M) \equiv 0 \pmod{p*q} \text{ oder } M^{e*d} \equiv M \pmod{p*q} \text{ oder } M^{e*d} \equiv M \pmod{n}.$$

□

### Bemerkung 1:

Man kann die 3 Schritte auch kürzer zusammenfassen, wenn man Satz 4.8.6 (Euler-Fermat) benutzt (also nicht den vereinfachten Satz, wo  $n = p$  gilt):

$$(M^e)^d \equiv M^{e*d} \equiv M^{(p-1)(q-1)*k+1} \equiv \underbrace{(M^{(p-1)(q-1)})^k}_{\equiv M^{J(n)} \equiv 1 \pmod{n}} * M \equiv 1^k * M \equiv M \pmod{n}.$$

### Bemerkung 2:

Beim Signieren werden die gleichen Operationen durchgeführt, aber zuerst mit dem geheimen Schlüssel  $d$ , und dann mit dem öffentlichen Schlüssel  $e$ . Das RSA-Verfahren ist auch für die Erstellung von digitalen Signaturen einsetzbar, weil gilt:

$$M \equiv (M^d)^e \pmod{n}.$$

## 4.11 Zur Sicherheit des RSA-Verfahrens<sup>52</sup>

Die Eignung des RSA-Verfahrens für digitale Signaturen und Verschlüsselung gerät immer wieder in die Diskussion, z.B. im Zusammenhang mit der Veröffentlichung aktueller Faktorisierungserfolge. Ungeachtet dessen ist das RSA-Verfahren seit seiner Veröffentlichung vor mehr als 20 Jahren unangefochtener De-facto-Standard (vgl. 7.1).

Die Sicherheit des RSA-Verfahrens basiert - wie die aller kryptographischen Verfahren - auf den folgenden 4 zentralen Säulen:

- der Komplexität des dem Problem zugrunde liegenden zahlentheoretischen Problems (hier der Faktorisierung großer Zahlen),
- der Wahl geeigneter Sicherheitsparameter (hier der Länge des Moduls  $n$ ),
- der geeigneten Anwendung des Verfahrens sowie der Schlüsselerzeugung und
- der korrekten Implementierung des Algorithmus.

Die Anwendung und Schlüsselerzeugung wird heute gut beherrscht. Die Implementierung ist auf Basis einer Langzahlarithmetik sehr einfach.

In den folgenden Abschnitten werden die ersten beiden Punkte näher untersucht.

### 4.11.1 Komplexität

Ein erfolgreiches Entschlüsseln oder eine Signaturfälschung — ohne Kenntnis des geheimen Schlüssels — erfordert, die  $e$ -te Wurzel mod  $n$  zu ziehen. Der geheime Schlüssel, nämlich die multiplikative Inverse zu  $e$  mod  $J(n)$ , kann leicht bestimmt werden, wenn die Eulersche Funktion  $J(n)$  bekannt ist.  $J(n)$  wiederum lässt sich aus den Primfaktoren von  $n$  berechnen. Das Brechen des RSA-Verfahrens kann daher nicht schwieriger sein als das Faktorisieren des Moduls  $n$ .

Das beste heute bekannte Verfahren ist eine Weiterentwicklung des ursprünglich für Zahlen mit einer bestimmten Darstellung (z.B. Fermatzahlen) entwickelten General Number Field Sieve (GNFS). Die Lösungskomplexität des Faktorisierungsproblems liegt damit asymptotisch bei

$$O(l) = e^{c \cdot (l \cdot \ln 2)^{1/3} \cdot (\ln(l \cdot \ln(2)))^{2/3} + o(l)}$$

Siehe:

A. Lenstra, H. Lenstra: *The development of the Number Field Sieve* [Lenstra1993].

Robert D. Silverman: *A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths* [Silverman2000].

Die Formel zeigt, dass das Faktorisierungsproblem zur Komplexitätsklasse der Probleme mit subexponentieller Berechnungskomplexität gehört (d.h. der Lösungsaufwand wächst asymptotisch nicht so stark wie  $e^l$  oder  $2^l$ , sondern echt schwächer, z. B. wie  $e^{\sqrt{l}}$ ). Diese Einordnung entspricht dem heutigen Kenntnisstand, sie bedeutet jedoch nicht, dass das Faktorisierungsproblem möglicherweise nicht auch mit (asymptotisch) polynomiellem Aufwand gelöst werden kann (s. 4.11.5).

---

<sup>52</sup>Große Teile des Kapitels 4.11 sind angelehnt an den Artikel „Vorzüge und Grenzen des RSA-Verfahrens“ von F. Bourseau, D. Fox und C. Thiel [Bourseau2002].

$O(l)$  gibt die Zahl der durchschnittlich erforderlichen Prozessor-Operationen abhängig von der Bitlänge  $l$  der zu faktorisierenden Zahl  $n$  an. Für den besten allgemeinen Faktorisierungsalgorithmus ist die Konstante  $c = (64/9)^{1/173} = 1,923$ .

Die umgekehrte Aussage, dass das RSA-Verfahren ausschließlich durch eine Faktorisierung von  $n$  gebrochen werden kann, ist bis heute nicht bewiesen. Zahlentheoretiker halten das “RSA-Problem” und das Faktorisierungsproblem für komplexitätstheoretisch äquivalent.

Siehe: *Handbook of Applied Cryptography* [Menezes2001].

#### 4.11.2 Sicherheitsparameter aufgrund neuer Algorithmen

##### Faktorisierungsalgorithmen<sup>53</sup>

Die Komplexität wird im wesentlichen von der Länge  $l$  des Moduls  $n$  bestimmt. Höhere Werte für diesen wesentlichen Parameter orientieren sich an den Möglichkeiten der aktuellen Faktorisierungsalgorithmen:

- 1994 wurde mit einer verteilten Implementierung des 1982 von Pomerance entwickelten Quadratic Sieve-Algorithmus (QS) nach knapp 8 Monaten der 1977 veröffentlichte 129-stellige RSA-Modul (428 bit) faktorisiert.

Siehe:

C. Pomerance: *The quadratic sieve factoring algorithm* [Pomerance1984].

- 1999 wurde mit dem von Buhler, Lenstra und Pomerance entwickelten General Number Field Sieve-Algorithmus (GNFS), der ab ca. 116 Dezimalstellen effizienter ist als QS, nach knapp 5 Monaten ein 155-stelliger Modul (512 bit) faktorisiert.

Siehe:

J.P. Buhler, H.W. Lenstra, C. Pomerance: *Factoring integers with the number field sieve* [Buhler1993].

- Ende 2009, also rund 10 Jahre später, wurde von Kleinjung etc. nach rund 2 1/2 Jahren ein 232-stelliger Modul (768 bit) faktorisiert.

Siehe:

T. Kleinjung, et. al.: *Factorization of a 768-bit RSA modulus* [Kleinjung2010].

Damit ist auch praktisch klar geworden, dass eine Modullänge von 768 bit keinen Schutz mehr vor Angreifern darstellt.

Details zu den Fortschritten bei der Faktorisierung seit 1999: siehe Kapitel 4.11.4.

##### Algorithmen zur Gitterbasenreduktion

Die Modullänge ist aber nicht der einzige Sicherheits-relevante Parameter. Neben Implementierungsanforderungen kommt es auch auf die Größen und die Größenverhältnisse der Parameter  $e$ ,  $d$  und  $n$  an.

Entsprechende Angriffe, die auf Gitterbasenreduktion beruhen, stellen für einfache RSA-Implementierungen eine reale Bedrohung dar. Diese Angriffe lassen sich in die folgenden vier Kategorien unterteilen:

<sup>52</sup>Mit dem Lernprogramm **ZT** können Sie einige der gängigen Faktorisierungsverfahren (Fermat, Pollard-Rho, Polard p-1, QS) vertiefen (siehe Lern-Kapitel 5.1-5.5, Seiten 1-15/15).

ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

- Angriffe auf sehr kleine öffentliche Schlüssel  $e$  (z.B.  $e = 3$ ).
- Angriffe auf relativ kurze geheime Exponenten  $d$  (z.B.  $d < n^{0,5}$ ).
- Faktorisierung des Moduls  $n$ , wenn einer der Faktoren  $p$  oder  $q$  *teilweise* bekannt ist.
- Angriffe, die voraussetzen, dass ein *Teil* des geheimen Schlüssels  $d$  bekannt ist.

Eine gute Übersicht zu diesen Angriffen findet sich in der Diplomarbeit *Analyse der Sicherheit des RSA-Algorithmus. Mögliche Angriffe, deren Einfluss auf sichere Implementierungen und ökonomische Konsequenzen* von Matthias Schneider [SchneiderM2004].

### 4.11.3 Vorhersagen zur Faktorisierung großer Zahlen

Seit 1980 wurden erhebliche Fortschritte gemacht. Abschätzungen über die zukünftige Entwicklung der Sicherheit unterschiedlicher RSA-Modullängen differieren und hängen von verschiedenen Annahmen ab:

- Entwicklung der Rechnergeschwindigkeit (Gesetz von Moore: Verdopplung der Rechnerleistung alle 18 Monate) und des Grid-Computing.
- Entwicklung neuer Algorithmen.

Selbst ohne neue Algorithmen wurden in den letzten Jahren durchschnittlich ca. 10 Bit mehr pro Jahr berechenbar. Größere Zahlen erfordern mit den heute bekannten Verfahren einen immer größeren Arbeitsspeicher für die Lösungsmatrix. Dieser Speicherbedarf wächst mit der Wurzel des Rechenzeitbedarfs, also ebenfalls subexponentiell. Da in den letzten Jahren der verfügbare Hauptspeicher ebenso wie die Rechengeschwindigkeit exponentiell gewachsen ist, dürfte hierdurch kein zusätzlicher Engpass entstehen.

Lenstra/Verheul [Lenstra1999] trafen eine Abschätzung für die Entwicklung sicherer Schlüssellängen (vergleiche Abbildung 7.1 in Kapitel 7.1).

In dem Artikel [Bourseau2002] veröffentlichte Dirk Fox<sup>54</sup> seine Prognose über einen annähernd linearen Verlauf der Faktorisierungserfolge unter Einbeziehung aller Einflussfaktoren: Pro Jahr kommen durchschnittlich 20 Bit dazu. Damit liegt er unter den optimistischeren Schätzungen von BSI und NIST.

Diese Prognose von Dirk Fox aus dem Jahr 2001 scheint sich anhand der neuesten Faktorisierungsrekorde von RSA-200 und RSA-768 (siehe Kapitel 4.11.4) zu bestätigen: Seine Schätzung für das Jahr 2005 mit 660 Bit hat die Bitlänge von RSA-200 nahezu auf den Punkt getroffen (vergleiche Abbildung 4.1).

Hält die Prognose, dann ist die Faktorisierung eines RSA-Moduls der Länge 1024 bit im Jahre 2020 zu erwarten.

<sup>54</sup>Seine Firma Secorvo GmbH hat auch eine Stellungnahme zur Schlüssellängenempfehlung des BSI für den Bundesanzeiger abgegeben. Kapitel 2.3.1 dieser Stellungnahme geht kompetent und verständlich auf RSA-Sicherheit ein (existiert nur in Deutsch):  
<http://www.secorvo.de/publikat/stellungnahme-algorithmenempfehlung-020307.pdf>

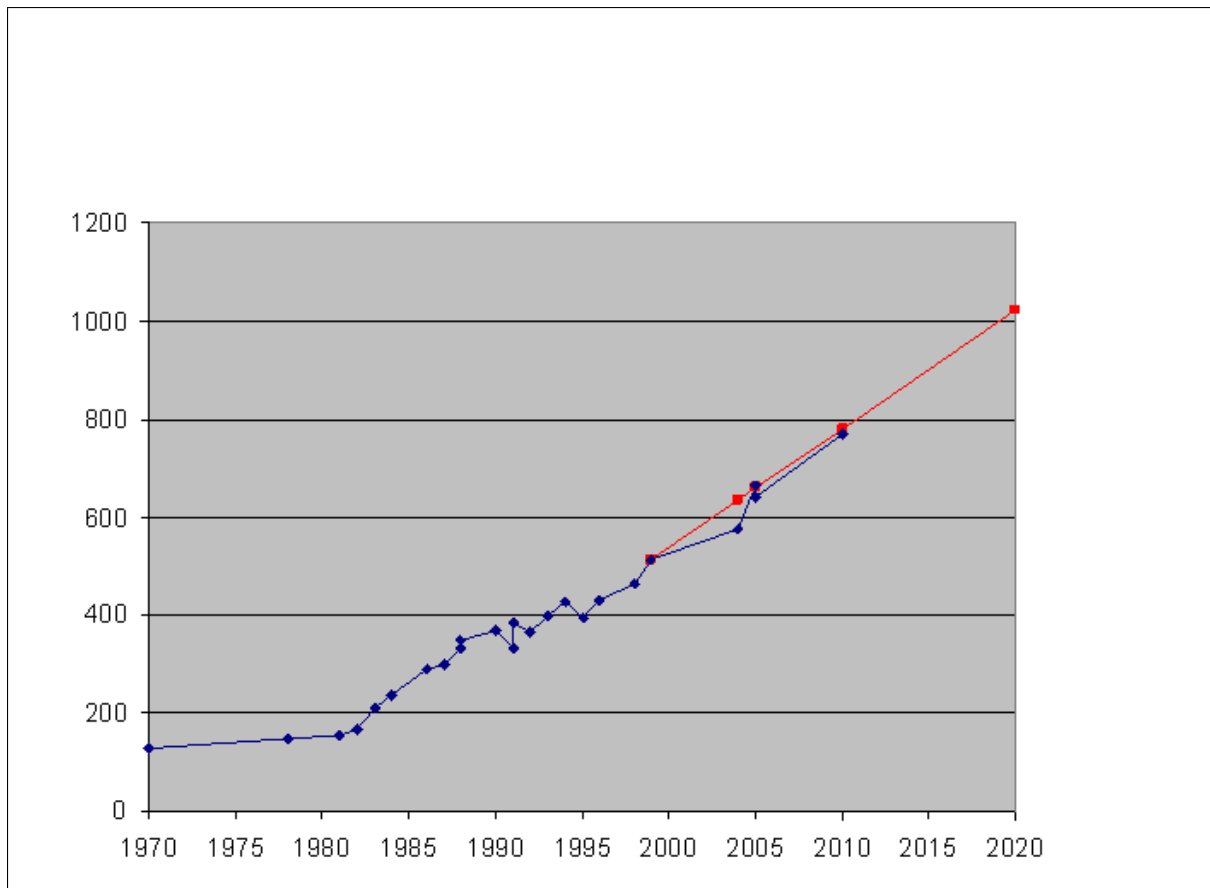


Abbildung 4.1: Vergleich der publizierten Faktorisierungserfolge (blau) mit der prognostizierten Entwicklung (rot; Quelle Fox 2001)

*Hermann Hesse*<sup>55</sup>:

Damit das Mögliche entsteht, muss immer wieder das Unmögliche versucht werden.

#### 4.11.4 Status der Faktorisierung von konkreten großen Zahlen

Ausführliche Übersichten über die Rekorde im Faktorisieren zusammengesetzter Zahlen mit unterschiedlichen Methoden finden sich auf den folgenden Webseiten:

<http://www.crypto-world.com>

[http://www.tutorgig.com/ed/RSA\\_number](http://www.tutorgig.com/ed/RSA_number) The RSA Factoring Challenge

[http://en.wikipedia.org/wiki/Integer\\_factorization\\_records](http://en.wikipedia.org/wiki/Integer_factorization_records)

[http://en.wikipedia.org/wiki/RSA\\_Factoring\\_Challenge](http://en.wikipedia.org/wiki/RSA_Factoring_Challenge)

Der aktuelle Rekord (Stand Jan. 2010) mit der GNFS-Methode (General Number Field Sieve) liegt in der Zerlegung einer allgemeinen 232-stelligen Dezimalzahl in ihre beiden Primfaktoren.

Die letzten Rekorde<sup>56</sup> mit Faktorisierungsverfahren für zusammengesetzte Zahlen sind in der folgenden Tabelle 4.10 aufgeführt:

	Dezimalstellen	Binärstellen	Faktorisiert am	Faktorisiert von
RSA-768	232	768	Dez 2010	Thorsten Kleinjung et al.
RSA-200	200	663	Mai 2005	Jens Franke et al.
RSA-640 <sup>57</sup>	193	640	Nov 2005	Jens Franke et al.
RSA-576	174	576	Dez 2003	Jens Franke et al.
RSA-160	160	530	Apr 2003	Jens Franke et al.
RSA-155	155	512	Aug 1999	Herman te Riele et al.
...				
C307	307	1017	Mai 2007	Jens Franke et al.
C176	176	583	Mai 2005	Kazumaro Aoki et al.
C158	158	523	Jan 2002	Jens Franke et al.

Tabelle 4.10: Die derzeitigen Faktorisierungsrekorde (Stand Jan. 2010)

<sup>55</sup>Hermann Hesse, deutsch-schweizerischer Schriftsteller und Nobelpreisträger, 02.07.1877–09.08.1962.

<sup>56</sup>Die „RSA-Zahlen“ sind große semiprime Zahlen (d.h. Zahlen, die genau aus 2 Primfaktoren bestehen). Sie wurden von der Firma RSA Security generiert und veröffentlicht: Im Wettbewerb „RSA Factoring Challenge“ wurden die Primfaktoren dieser Zahlen gesucht.

Siehe <http://www.rsa.com/rsalabs/node.asp?id=2092>.

Die RSA-Laboratorien schreiben ihre Challenges schon seit Anfang der 90er-Jahre aus. In der ersten RSA-Factoring-Challenge wurden die Zahlen, von RSA-100 bis RSA-500, gemäß der Anzahl ihrer Dezimalstellen benannt; die zweite RSA-Factoring-Challenge benannte die Zahlen anhand der Anzahl ihrer Binärstellen. Innerhalb des zweiten Wettbewerbs wurden Geldpreise für die erfolgreiche Faktorisierung von RSA-576 bis RSA-2048 ausgelobt (RSA-576, RSA-640 etc. in 64-er Schritten — Die Zahl RSA-617 bildet eine Ausnahme, da sie vor der Änderung des Namensschemas erzeugt wurde). Leider beendete RSA Inc. die Challenge vorzeitig und zog die Preise zurück.

Die „C-Zahlen“ stammen aus dem Cunningham-Projekt: <http://www.cerias.purdue.edu/homes/ssw/cun/>. Diese sind Faktoren großer Mersennezahlen, die eine ganz spezielle Struktur haben. Dies macht es um Größenordnungen einfacher, sie zu faktorisieren, als Moduli gleicher Längen, die man für RSA erstellt.

Im Folgenden werden diese letzten Rekorde etwas ausführlicher erläutert. Die beiden dabei benutzten Methoden GNFS und SNFS werden kurz z.B. auf den ff. Webseiten dargestellt:

[http://en.wikipedia.org/wiki/Special\\_number\\_field\\_sieve](http://en.wikipedia.org/wiki/Special_number_field_sieve)

[http://en.wikipedia.org/wiki/General\\_number\\_field\\_sieve](http://en.wikipedia.org/wiki/General_number_field_sieve)

## RSA-155

Am 22. August 1999 fanden niederländische Forscher die Lösung dieser RSA-Challenge. Sie zerlegten eine 155-stellige Zahl in ihre beiden 78-stelligen Primfaktoren (vergleiche Kapitel 4.11.2). Mit der 512 Bit-Zahl RSA-155 war eine *magische* Grenze erreicht.

## C158

Am 18. Januar 2002 zerlegten Forscher der Universität Bonn<sup>58</sup> mit der GNFS-Methode (General Number Field Sieve) eine 158-stellige Dezimalzahl in ihre beiden Primfaktoren (diese haben 73 und 86 Dezimalstellen).

Dieser Rekord fand deutlich weniger Aufmerksamkeit in der Presse als die Lösung von RSA-155.

Die Aufgabe der Bonner Wissenschaftler entsprang auch nicht einer Challenge, sondern die Aufgabe war, die letzten Primfaktoren der Zahl  $2^{953} + 1$  zu finden (siehe "Wanted List" des Cunningham-Projekts<sup>59</sup>).

Die 6 kleineren, schon vorher gefundenen Primfaktoren dieser Zahl waren:

3, 1907, 425796183929,  
1624700279478894385598779655842584377,  
3802306738549441324432139091271828121 und  
128064886830166671444802576129115872060027.

Die drei kleinsten Faktoren können leicht<sup>60</sup> bestimmt werden. Die nächsten drei Primfaktoren wurden von P. Zimmerman<sup>61</sup>, T. Grandlund<sup>62</sup> und R. Harley in den Jahren 1999 und 2000 mit der Methode der Elliptischen Kurven gefunden.

Als letzter Faktor blieb der sogenannte Teiler „C158“, von dem man bis dahin wusste, dass er zusammengesetzt ist, aber man kannte seine Primfaktoren nicht (die folgenden drei Zeilen sind eine einzige Zahl):

39505874583265144526419767800614481996020776460304936  
45413937605157935562652945068360972784246821953509354  
4305870490251995655335710209799226484977949442955603

<sup>57</sup>Eine Arbeitsgruppe des BSI löste die mit 20.000 US-Dollar dotierte Challenge RSA-640 mit Hilfe der GNFS-Methode. Die Forscher benötigten für die Zerlegung der Zahl in ihre beiden 320 Bit langen Primfaktoren rund fünf Monate Rechenzeit.

Die Forscher um Prof. Jens Franke (von der Universität Bonn, dem BSI und dem CWI) waren nicht auf die Geldpreise aus, sondern wollten die Grenzen der Forschung ausdehnen. Dadurch werden Aussagen über notwendige Minimallängen für sichere RSA-Moduli fundierter.

Siehe <http://www.heise.de/newsticker/meldung/print/65957>

<sup>58</sup><http://www.uni-bonn.de/Aktuelles/Pressemitteilungen/pm02/pm035-02.html>

<sup>59</sup>Cunningham-Projekt: <http://www.cerias.purdue.edu/homes/ssw/cun/>

<sup>60</sup>Z.B. mit CrypTool über das Menü **Einzelverfahren \ RSA-Kryptosystem \ Faktorisieren einer Zahl**.

In sinnvoller Zeit zerlegt CrypTool Zahlen bis 250 Bit Länge. Zahlen größer als 1024 Bit werden zur Zeit von CrypTool nicht angenommen.

<sup>61</sup><http://www.loria.fr/zimmerma/ecmnet>

<sup>62</sup><http://www.swox.se/gmp/>



Die Faktorisierung von C158 ergab die beiden 73- und 86-stelligen Primfaktoren:

3388495837466721394368393204672181522815830368604993048084925840555281177

und

1165882340667125990314837655838327081813101

2258146392600439520994131344334162924536139.

Damit wurde die Zahl  $2^{953} + 1$  vollständig in ihre 8 Primfaktoren zerlegt.

Verweise:

<http://www.loria.fr/~zimmerma/records/gnfs158>

<http://www.crypto-world.com/FactorRecords.html>

<http://www.crypto-world.com/announcements/c158.txt>

## RSA-160

Am 1. April 2003 zerlegten Forscher der Universität Bonn<sup>63</sup> mit der GNFS-Methode (General Number Field Sieve) eine 160-stellige Zahl in ihre beiden Primfaktoren (diese haben jeweils 80 Dezimalstellen).

Die Berechnungen dazu fanden auch im Bundesamt für Sicherheit in der Informationstechnik (BSI) in Bonn statt<sup>64</sup>.

Die 160-stellige Dezimalzahl stammt von der alten Challenge-Liste von RSADSI. Diese wurde nach der Faktorisierung von RSA-155 (RSA512) zurückgezogen. Die Primfaktoren von RSA-160 waren aber nicht bekannt. Deshalb ist dieser Rekord von Prof. Frankes Team immer noch die Lösung einer alten Challenge, für die es aber von RSADSI kein Geld gibt.

Die zusammengesetzte Zahl „RSA-160“ lautet (die folgenden drei Zeilen sind eine einzige Zahl):

215274110271888970189601520131282542925777358884567598017049

767677813314521885913567301105977349105960249790711158521430

2079314665202840140619946994927570407753

Die Faktorisierung von RSA-160 ergab die beiden Primfaktoren:

$p = 45427892858481394071686190649738831$

656137145778469793250959984709250004157335359

und

$q = 47388090603832016196633832303788951$

973268922921040957944741354648812028493909367

Die Berechnungen erfolgten zwischen Dezember 2002 und April 2003.

---

<sup>63</sup><http://www.loria.fr/~zimmerma/records/rsa160>  
<http://www.loria.fr/~zimmerma/records/factor.html>  
<http://www.crypto-world.com/FactorWorld.html>

<sup>64</sup>Das BSI erstellt jedes Jahr ein Papier über die Eignung von Kryptoalgorithmen, mit denen Signaturen erzeugt werden können, die den Vorgaben des deutschen Signaturgesetzes genügen. Bei dieser Erstellung werden Experten aus Wirtschaft und Wissenschaft beteiligt. Um die Eignung von Signaturverfahren zu beurteilen, deren Schwierigkeit auf dem Faktorisierungsproblem beruht, kooperiert das BSI auch mit Forschern der Universität Bonn. Weitere Informationen zu Kryptoalgorithmen finden Sie auf den BSI-Internetseiten unter: <http://www.bsi.bund.de/esig/basics/techbas/krypto/index.htm>

## RSA-200

Am 9. Mai 2005 meldete die Forschergruppe von Prof. Jens Franke der Universität Bonn<sup>65</sup>, dass sie gemeinsam mit Kollegen des Amsterdam Centrum voor Wiskunde en Informatica einen neuen Weltrekord im Faktorisieren aufstellten.

Sie zerlegten mit der GNFS-Methode (General Number Field Sieve) eine 200-stellige Zahl in ihre beiden Primfaktoren (diese haben jeweils 100 Dezimalstellen).

Die zusammengesetzte Zahl „RSA-200“ lautet (die folgenden drei Zeilen sind eine einzige Zahl):

```
2799783391122132787082946763872260162107044678695542853756000992932
6128400107609345671052955360856061822351910951365788637105954482006
576775098580557613579098734950144178863178946295187237869221823983
```

Die Faktorisierung von RSA-200 ergab die beiden Primfaktoren:

$$p = 35324619344027701212726049781984643686711974001976 \\ 25023649303468776121253679423200058547956528088349$$

und

$$q = 79258699544783330333470858414800596877379758573642 \\ 19960734330341455767872818152135381409304740185467$$

Die Berechnungen erfolgten zwischen Dezember 2003 und Mai 2005. Die Faktorisierung durch die Gruppe um Bahr, Böhm, Franke, Kleinjung, Montgomery und te Riele hatte also knapp 17 Monate gedauert. Der Rechenaufwand lag bei umgerechnet etwa 120.000 MIPS-Jahren<sup>66</sup>.

## RSA-768

Am 12. Dezember 2009 meldete die Forschergruppe um Prof. Thorsten Kleinjung<sup>67</sup>, dass sie eine 232-stellige Zahl in ihre beiden Primfaktoren zerlegten (diese haben jeweils 116 Dezimalstellen). Sie benutzten dazu die GNFS-Methode (General Number Field Sieve) in einer Art, dass vor dem Matrix-Schritt auf mehreren hundert Rechnern „Oversieving“ betrieben wurde.

Die zusammengesetzte Zahl „RSA-768“ lautet (die folgenden drei Zeilen sind eine einzige Zahl):

```
123018668453011775513049495838496272077285356959533479219732245215172640050726
365751874520219978646938995647494277406384592519255732630345373154826850791702
6122142913461670429214311602221240479274737794080665351419597459856902143413
```

Die Faktorisierung von RSA-768 ergab die beiden Primfaktoren (je 384 bit):

$$p = 3347807169895689878604416984821269081770479498371376856891 \\ 2431388982883793878002287614711652531743087737814467999489$$

<sup>65</sup><http://www.loria.fr/~zimmerma/records/rsa200>

<sup>66</sup>Ein MIPS-Jahr (MY) ist die Anzahl von Operationen, die eine Maschine, welche eine Million Integeroperationen pro Sekunde (MIPS) ausführt, in einem Jahr bewältigt. Zur Illustration: ein INTEL Pentium 100 Prozessor hat etwa 50 MIPS. Für die Zerlegung eines 2048-Bit-Moduls bräuchte man ca.  $8,5 \cdot 10^{40}$  MY.

<sup>67</sup><http://eprint.iacr.org/2010/006.pdf> [Kleinjung2010]

und

$q = 3674604366679959042824463379962795263227915816434308764267$   
 $6032283815739666511279233373417143396810270092798736308917$

Die Berechnungen dauerten 2 1/2 Jahre<sup>68</sup>.

### C307 / M1039

Im Mai 2007 meldeten Prof. Franke, Prof. Kleinjung (von der Universität Bonn), das japanische Telekommunikationsunternehmen NTT und Prof. Arjen Lenstra von der Polytechnischen Hochschule in Lausanne, dass sie mit der SNFS-Methode (Special Number Field Sieve) innerhalb von 11 Monaten eine 307-stellige Dezimalzahl in ihre beiden Primfaktoren zerlegten (diese haben 80 und 227 Dezimalstellen).

Die Aufgabe der Wissenschaftler entsprang nicht einer Challenge, sondern die Aufgabe war, die letzten Primfaktoren der Mersenne-Zahl  $2^{1039} + 1$  zu finden (siehe „Wanted List“ des Cunningham-Projekts<sup>69</sup>).

Die Zahl  $2^{1039} - 1$  besteht aus 3 Primfaktoren: Der kleinste Faktor  $p_7 = 5080711$  war schon länger bekannt.<sup>70</sup>

Zur vollständigen Faktorisierung musste der zweite Faktor (Koteiler) „C307“ zerlegt werden: Bis dahin wusste man nur, dass er zusammengesetzt ist, aber man kannte weder die Anzahl seiner Primfaktoren, noch die Primfaktoren selbst. Die folgenden fünf Zeilen sind eine einzige Zahl:

C307 = 1159420574072573064369807148876894640753899791702017724986868353538  
8224838599667566080006095408005179472053993261230204874402860435302  
8619141014409345351233471273967988850226307575280937916602855510550  
0425810771176177610094137970787973806187008437777186828680889844712  
822002935201806074755451541370711023817

<sup>68</sup>Dies war ein „academic effort“ – Organisationen mit besserer Ressourcen-Ausstattung könnten es deutlich schneller durchführen.

<sup>69</sup>Cunningham-Projekt: <http://www.cerias.purdue.edu/homes/ssw/cun/>  
Cunningham-Tabelle: <http://homes.cerias.purdue.edu/~ssw/cun/pmain1206>  
Die Zahlen in der Cunningham-Tabelle werden folgendermaßen geschrieben:

„(2,n)–“ bedeutet  $2^n - 1$ ; „(2,n)+“ bedeutet  $2^n + 1$ .

Um die Größenordnung einer Zerlegung anzudeuten schreibt man  $p < n$  oder  $c < n$ , wobei „n“ die Anzahl der Dezimalstellen ist und „p“ und „c“ bedeuten, dass die Zahl eine Primzahl oder eine zusammengesetzte Zahl ist.

$2^{1039} - 1 = p_7 * c_{307} = p_7 * p_{80} * p_{227}$

Genauer erklärt wird dies auf der CUN-Seite folgendermaßen:

„2,651+ means  $2^{651} + 1$  and the size (c209 means 209 decimal digits) of the number which was factored. Then come the new factor(s), the discoverer and the method used. Recently, only the multiple polynomial quadratic sieve (ppmpqs), the elliptic curve method (ecm) and the number field sieve (nfs) have been used. ‘hmpqs’ stands for hypercube multiple polynomial quadratic sieve. Under ‘new factors’, ‘p90’ means a 90-digit prime and ‘c201’ is a 201-digit composite number.“

<sup>70</sup>Er kann mit CrypTool über das Menü **Einzelverfahren \ RSA-Kryptosystem \ Faktorisieren einer Zahl** gefunden werden — mit den Algorithmen von Brent, Williams oder Lenstra, mit denen man gut „relativ“ kleine Faktoren abspalten kann.

In sinnvoller Zeit zerlegt CrypTool vollständig Zahlen bis 250 Bit Länge.

Die Faktorisierung von C307 ergab die beiden 80- und 227-stelligen Primfaktoren:

$$p_{80} = 558536666199362912607492046583159449686465270184 \\ 88637648010052346319853288374753$$

und

$$p_{227} = 207581819464423827645704813703594695162939708007395209881208 \\ 387037927290903246793823431438841448348825340533447691122230 \\ 281583276965253760914101891052419938993341097116243589620659 \\ 72167481161749004803659735573409253205425523689.$$

Damit wurde die Zahl  $2^{1039} - 1$  vollständig in ihre 3 Primfaktoren zerlegt.

Verweise:

<http://www.loria.fr/~zimmerma/records/21039->  
<http://www.crypto-world.com/announcements/m1039.txt>  
<http://www.crypto-world.com/FactorAnnouncements.html>  
[http://www1.uni-bonn.de/pressDB/jsp/pressemitteilungsdetails.jsp?  
detailjahr=2007&detail=160](http://www1.uni-bonn.de/pressDB/jsp/pressemitteilungsdetails.jsp?detailjahr=2007&detail=160)

### Größenordnung faktorisierter Zahlen im Vergleich zu auf Primalität getesteter Zahlen

Wie man sieht, sind die größten (aus 2 Primfaktoren) zusammengesetzten Zahlen, die man faktorisieren kann, deutlich kleiner als die Zahlen mit einer speziellen Struktur, für die Primzahltests in der Lage sind, Aussagen über ihre Primalität zu treffen (siehe Kapitel 3.4, 3.5 und 3.6).

Länge in Bit der derzeitigen Weltrekorde:

$$[RSA-768-Zahl] \longleftrightarrow [47. \text{ bekannte Mersenne Primzahl}] \\ 768 \longleftrightarrow 43.112.609$$

#### 4.11.5 Weitere aktuelle Forschungsergebnisse zu Primzahlen und Faktorisierung

Primzahlen sind Teil vieler hochaktueller Forschungsgebiete der Zahlentheorie. Die Fortschritte bei der Faktorisierung sind groß als noch vor 5 Jahren geschätzt – sie gehen nicht nur auf das Konto schnellerer Rechner, sondern sie sind auch in neuen Erkenntnissen begründet.

Die Sicherheit des RSA-Algorithmus basiert auf der empirischen Beobachtung, dass die Faktorisierung großer ganzer Zahlen ein schwieriges Problem ist. Besteht wie beim RSA-Algorithmus der zugrunde liegende Modul  $n$  aus dem Produkt zweier großer Primzahlen  $p, q$  (typische Längen:  $p, q$  500 – 600 bit,  $n$  1024 bit), so lässt sich  $n = pq$  aus  $p$  und  $q$  leicht bestimmen, jedoch ist es mit

den bisher bekannten Faktorisierungsalgorithmen nicht möglich,  $p, q$  aus  $n$  zu gewinnen. Nur mit Kenntnis von  $p$  und  $q$  lässt sich jedoch der private aus dem öffentlichen Schlüssel ermitteln.

Die Entdeckung eines Algorithmus zur effizienten Faktorisierung von Produkten  $n = pq$  großer Primzahlen würde daher den RSA-Algorithmus wesentlich beeinträchtigen. Je nach Effizienz der Faktorisierung im Vergleich zur Erzeugung von  $p, q, n$  müsste der verwendete Modul  $n$  (z.Zt. 1024 bit) erheblich vergrößert oder — im Extremfall — auf den Einsatz des RSA ganz verzichtet werden.

## Das Papier von Bernstein und seine Auswirkungen auf die Sicherheit des RSA-Algorithmus

Die im November 2001 veröffentlichte Arbeit “Circuits for integer factorization: a proposal” (siehe <http://cr.yp.to/djb.html>) von D.J. Bernstein [Bernstein2001] behandelt das Problem der Faktorisierung großer Zahlen. Die Kernaussage des Papers besteht darin, dass es möglich ist, die Implementierung des General Number Field Sieve-Algorithmus (GNFS) so zu verbessern, dass mit gleichem Aufwand wie bisher Zahlen mit 3-mal größerer Stellenzahl (Bit-Länge) faktorisiert werden können.

Wesentlich bei der Interpretation des Resultats ist die Definition des Aufwandes: Als Aufwand wird das Produkt von benötigter Rechenzeit und Kosten der Maschine (insbesondere des verwendeten Speicherplatzes) angesetzt. Zentral für das Ergebnis des Papiers ist die Beobachtung, dass ein wesentlicher Teil der Faktorisierung auf Sortierung zurückgeführt werden kann und mit dem Schimmlerschen Sortierschema ein Algorithmus zur Verfügung steht, der sich besonders gut für den Einsatz von Parallelrechnern eignet. Am Ende des Abschnittes 3 gibt Bernstein konkret an, dass die Verwendung von  $m^2$  Parallelrechnern mit jeweils gleicher Menge an Speicherplatz mit Kosten in der Größenordnung von  $m^2$  einhergeht — genau so wie ein einzelner Rechner mit  $m^2$  Speicherzellen. Der Parallelrechner bewältigt die Sortierung von  $m^2$  Zahlen jedoch (unter Verwendung der o. g. Sortiervfahrens) in Zeit proportional zu  $m$ , wohingegen der Einprozessorrechner Zeit proportional  $m^2$  benötigt. Verringert man daher den verwendeten Speicherplatz und erhöht — bei insgesamt gleich bleibenden Kosten — die Anzahl der Prozessoren entsprechend, verringert sich die benötigte Zeit um die Größenordnung  $1/m$ . In Abschnitt 5 wird ferner angeführt, dass der massive Einsatz der parallelisierten Elliptic Curve-Methode von Lenstra die Kosten der Faktorisierung ebenfalls um eine Größenordnung verringert (ein Suchalgorithmus hat dann quadratische statt kubische Kosten). Alle Ergebnisse von Bernstein gelten nur asymptotisch für große Zahlen  $n$ . Leider liegen keine Abschätzungen über den Fehlerterm, d.h. die Abweichung der tatsächlichen Zeit von dem asymptotischen Wert, vor — ein Mangel, den auch Bernstein in seinem Papier erwähnt. Daher kann zur Zeit keine Aussage getroffen werden, ob die Kosten (im Sinne der Bernsteinschen Definition) bei der Faktorisierung z. Zt. verwendeter RSA-Zahlen (1024–2048 bit) bereits signifikant sinken würden.

Zusammenfassend lässt sich sagen, dass der Ansatz von Bernstein durchaus innovativ ist. Da die Verbesserung der Rechenzeit unter gleichbleibenden Kosten durch einen massiven Einsatz von Parallelrechnern erkaufte wird, stellt sich die Frage nach der praktischen Relevanz. Auch wenn formal der Einsatz von einem Rechner über 1 sec und 1.000.000 Rechnern für je 1/1.000.000 sec dieselben Kosten erzeugen mag, ist die Parallelschaltung von 1.000.000 Rechnern praktisch nicht (oder nur unter immensen Fixkosten, insbesondere für die Vernetzung der Prozessoren) zu realisieren. Solche Fixkosten werden aber nicht in Ansatz gebracht. Die Verwendung verteilter Ansätze (distributed computing) über ein großes Netzwerk könnte einen Ausweg bieten. Auch hier müssten Zeiten und Kosten für Datenübertragung einkalkuliert werden.

Solange noch keine (kostengünstige) Hardware oder verteilte Ansätze entwickelt wurden, die

auf dem Bernsteinschen Prinzip basieren, besteht noch keine akute Gefährdung des RSA. Es bleibt zu klären, ab welchen Größenordnungen von  $n$  die Asymptotik greift.

Arjen Lenstra, Adi Shamir et. al. haben das Bernstein-Paper analysiert [Lenstra2002]. Als Ergebnis kommen Sie zu einer Bitlängen-Verbesserung der Faktorisierung um den Faktor 1.17 (anstatt Faktor 3 wie von Bernstein erwartet).

Die Zusammenfassung ihres Papers “Analysis of Bernstein’s Factorization Circuit” lautet:

“... Bernstein proposed a circuit-based implementation of the matrix step of the number field sieve factorization algorithm. We show that under the non-standard cost function used in [1], these circuits indeed offer an asymptotic improvement over other methods but to a lesser degree than previously claimed: for a given cost, the new method can factor integers that are 1.17 times larger (rather than 3.01). We also propose an improved circuit design based on a new mesh routing algorithm, and show that for factorization of 1024-bit integers the matrix step can, under an optimistic assumption about the matrix size, be completed within a day by a device that costs a few thousand dollars. We conclude that from a practical standpoint, the security of RSA relies exclusively on the hardness of the relation collection step of the number field sieve.”

Auch RSA Security<sup>71</sup> kommt in ihrer Analyse der Bernstein-Arbeit [RSA Security 2002] vom 8. April 2002 erwartungsgemäß zu dem Ergebnis, dass RSA weiterhin als ungebrochen betrachtet werden kann.

Die Diskussion ist weiterhin im Gang.

Zum Zeitpunkt der Erstellung dieses Absatzes (Juni 2002) war nichts darüber bekannt, inwieweit die im Bernstein-Papier vorgeschlagenen theoretischen Ansätze realisiert wurden oder wieweit die Finanzierung seines Forschungsprojektes ist.

Verweise:

<http://cr.yp.to/djb.html>

<http://www.counterpane.com/crypto-gram-0203.html#6>

<http://www.math.uic.edu>

## Das TWIRL-Device

Im Januar 2003 veröffentlichten Adi Shamir und Eran Tromer vom Weizmann Institute of Science den vorläufigen Draft “*Factoring Large Numbers with the TWIRL Device*”, in dem deutliche Bedenken gegen RSA-Schlüssellängen unter 1024 begründet werden [Shamir2003].

Das Abstract fasst ihre Ergebnisse folgendermaßen zusammen: “The security of the RSA cryptosystem depends on the difficulty of factoring large integers. The best current factoring algorithm is the Number Field Sieve (NFS), and its most difficult part is the sieving step. In 1999 a large distributed computation involving thousands of workstations working for many months managed to factor a 512-bit RSA key, but 1024-bit keys were believed to be safe for the next 15-20 years. In this paper we describe a new hardware implementation of the NFS sieving step ... which is 3-4 orders of magnitude more cost effective than the best previously published designs ... . Based on a detailed analysis of all the critical components (but without an actual implementation), we believe that the NFS sieving step for 1024-bit RSA keys can be completed in less than a year by a \$10M device, and that the NFS sieving step for 512-bit RSA keys can be completed in less than ten minutes by a \$10K device. Coupled with recent results about the difficulty of the NFS matrix step ... this raises some concerns about the security of these key

---

<sup>71</sup><http://www.rsasecurity.com/>

sizes.”

Eine ausführliche Fassung findet sich auch in dem Artikel der beiden Autoren in den RSA Laboratories CryptoBytes [Shamir2003a].

Eine sehr gute Erläuterung, wie der Angriff mit dem Generalized Number Field Sieve (GNFS) funktioniert und welche Fortschritte sich ergaben, findet sich in dem 3-seitigen Artikel in der DuD-Ausgabe Juni/2003 [Weis2003]. Beim GNFS können 2 grundlegende Schritte unterschieden werden: der Siebungsschritt (Relationen sammeln) und die Matrix-Reduktion. Auch wenn der Siebungsschritt hochgradig parallelisierbar ist, so dominiert er doch den Gesamtrechenaufwand. Bisher haben Shamir und Tromer noch kein TWIRL-Device gebaut, jedoch ist der dafür geschätzte Aufwand von 10 bis 50 Millionen Euro, um eine 1024-Bit Zahl in einem Jahr zu faktorisieren für Geheimdienste oder große kriminelle Organisationen keineswegs prohibitiv, denn die „Kosten für einen einzigen Spionagesatelliten schätzt man z.B. auf mehrere Milliarden USD“. Die Autoren empfehlen deshalb konkret, möglichst rasch sensible, bisher benutzte RSA-, Diffie-Hellman- oder ElGamal-Schlüssel von bis zu 1024 Bit zu wechseln und Schlüssel von mindestens 2048 Bit Länge einzusetzen. Auch für die geplante T CPA/Palladium-Hardware werden 2048-Bit RSA-Schlüssel verwendet!

Damit erscheinen die aktuellen Empfehlungen des BSI, auf längere RSA-Schlüssellängen umzustellen, mehr als gerechtfertigt.

### **„Primes in P“: Testen auf Primalität ist polynomial**

Im August 2002 veröffentlichten die drei indischen Forscher M. Agrawal, N. Kayal und N. Saxena ihr Paper „*PRIMES in P*“ über einen neuen Primzahltest-Algorithmus, genannt AKS [Agrawal2002]. Sie entdeckten einen polynomialen deterministischen Algorithmus, um zu entscheiden, ob eine gegebene Zahl prim ist oder nicht.

Die Bedeutung dieser Entdeckung liegt darin, dass sie Zahlentheoretiker mit neuen Einsichten und Möglichkeiten für die weitere Forschung versorgt. Viele Menschen haben im Lauf der Jahrhunderte nach einem polynomialen Primzahltest gesucht, so dass dieses Ergebnis einen theoretischen Durchbruch darstellt. Es zeigt sich immer wieder, dass aus schon lange bekannten Fakten neue Ergebnisse generiert werden können.

Aber selbst die Autoren merken an, dass andere bekannte Algorithmen (z.B. ECPP) schneller sein können. Der neue Algorithmus funktioniert für alle positiven ganzen Zahlen. Dagegen verwendet das GIMPS-Projekt den Lucas-Lehmer-Primzahltest, der besondere Eigenschaften der Mersennezahlen ausnutzt. Dadurch ist der Lucas-Lehmer-Test viel schneller und erlaubt, Zahlen mit Millionen von Stellen zu testen, während generische Algorithmen auf Zahlen mit einigen tausend Stellen beschränkt sind. Nachteil der bisherigen schnellen Verfahren ist, dass sie probabilistisch sind, also ihr Ergebnis höchstwahrscheinlich, aber nicht ganz sicher ist.

Aktuelle Forschungsergebnisse dazu finden sich z.B. auf:

<http://www.mersenne.org/>

<http://fatphil.org/math/aks/> Originaltext in Englisch

<http://ls2-www.cs.uni-dortmund.de/lehre/winter200203/kt/material/primes.ps>

Gute Erläuterung in Deutsch von Thomas Hofmeister.

Joanne K. Rowling<sup>72</sup>:

Viel mehr als unsere Fähigkeiten sind es unsere Entscheidungen ..., die zeigen, wer wir wirklich sind.

## 4.12 Anwendungen asymmetrischer Kryptographie mit Zahlenbeispielen

In der modernen Kryptographie werden die Ergebnisse der modularen Arithmetik extensiv angewandt. Hier werden exemplarisch einige wenige Beispiele aus der Kryptographie mit kleinen<sup>73</sup> Zahlen vorgestellt.

Die Chiffrierung eines Textes besteht darin, dass man aus einer Zeichenkette (Zahl) durch Anwenden einer Funktion (mathematische Operationen) eine andere Zahl erzeugt. Dechiffrieren heißt, diese Funktion umzukehren: aus dem Zerrbild, das die Funktion aus dem Klartext gemacht hat, das Urbild wiederherzustellen. Beispielsweise könnte der Absender einer vertraulichen Nachricht zum Klartext  $M$  eine geheimzuhaltende Zahl, den Schlüssel  $S$ , addieren und dadurch den Chiffretext  $C$  erhalten:

$$C = M + S.$$

Durch Umkehren dieser Operation, das heißt durch Subtrahieren von  $S$ , kann der Empfänger den Klartext rekonstruieren:

$$M = C - S.$$

Das Addieren von  $S$  macht den Klartext zuverlässig unkenntlich. Gleichwohl ist diese Verschlüsselung sehr schwach; denn wenn ein Abhörer auch nur ein zusammengehöriges Paar von Klar- und Chiffretext in die Hände bekommt, kann er den Schlüssel berechnen

$$S = C - M,$$

und alle folgenden mit  $S$  verschlüsselten Nachrichten mitlesen.

Der wesentliche Grund ist, dass Subtrahieren eine ebenso einfache Operation ist wie Addieren.

### 4.12.1 Einwegfunktionen

Wenn der Schlüssel auch bei gleichzeitiger Kenntnis von Klar- und Chiffretext nicht ermittelbar sein soll, braucht man eine Funktion, die einerseits relativ einfach berechenbar ist - man will ja chiffrieren können. Andererseits soll ihre Umkehrung zwar existieren (sonst würde beim Chiffrieren Information verlorengehen), aber de facto unberechenbar sein.

Was sind denkbare Kandidaten für eine solche **Einwegfunktion**? Man könnte an die Stelle der Addition die Multiplikation setzen; aber schon Grundschüler wissen, dass deren Umkehrung, die Division, nur geringfügig mühsamer ist als die Multiplikation selbst. Man muss noch eine Stufe höher in der Hierarchie der Rechenarten gehen: Potenzieren ist immer noch eine relativ einfache Operation; aber ihre beiden Umkehrungen *Wurzelziehen* (finde  $b$  in der Gleichung  $a = b^c$ , wenn  $a$  und  $c$  bekannt sind) und *Logarithmieren* (in derselben Gleichung finde  $c$ , wenn

<sup>72</sup>Joanne K. Rowling, „Harry Potter und die Kammer des Schreckens“, Carlsen, (c) 1998, letztes Kapitel „Dobbys Belohnung“, S. 343, Dumbledore.

<sup>73</sup>„Klein“ bedeutet beim RSA-Verfahren, dass die Bitlängen der Zahlen sehr viel kleiner sind als 1024 Bit (das sind 308 Dezimalstellen). 1024 Bit gilt im Moment in der Praxis als Mindestlänge für einen sicheren Certification Authority-RSA-Modul.



$a$  und  $b$  bekannt sind) sind so kompliziert, dass ihre Ausführung in der Schule normalerweise nicht mehr gelehrt wird.

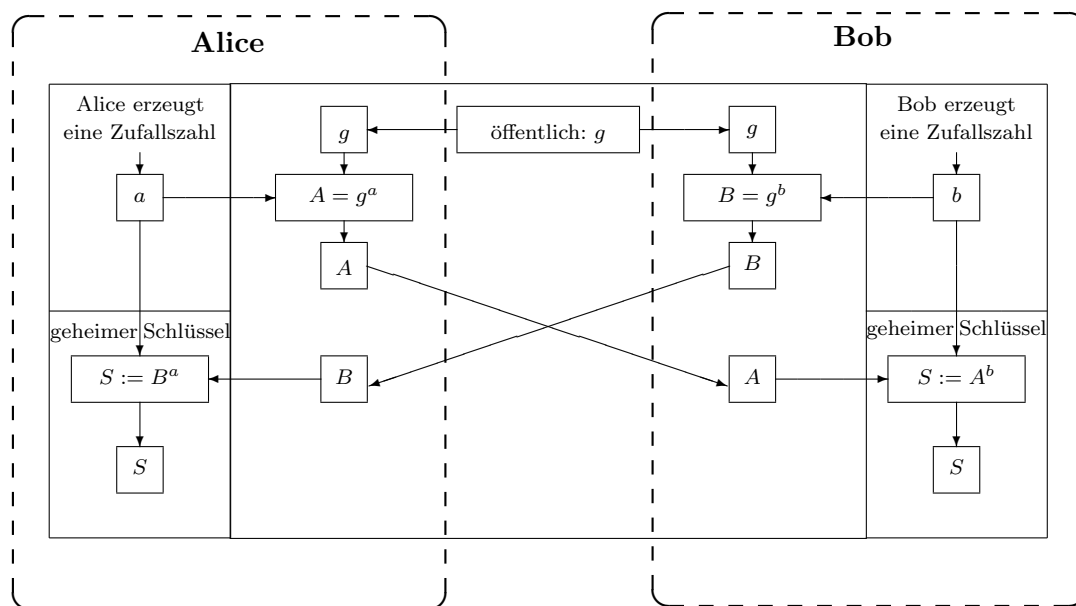
Während bei Addition und Multiplikation noch eine gewisse Struktur wiedererkennbar ist, wirbeln Potenzierung und Exponentiation alle Zahlen wild durcheinander: Wenn man einige wenige Funktionswerte kennt, weiß man (anders als bei Addition und Multiplikation) noch kaum etwas über die Funktion im ganzen.

#### 4.12.2 Das Diffie-Hellman Schlüsselaustausch-Protokoll (Key Exchange Protocol)

Das DH-Schlüsselaustauschprotokoll wurde 1976 in Stanford von Whitfield Diffie, Martin E. Hellman und Ralph Merkle erdacht<sup>74</sup>.

Eine Einwegfunktion dient Alice und Bob<sup>75</sup> dazu, sich einen Schlüssel  $S$ , den Sessionkey, für die nachfolgende Verständigung zu verschaffen. Dieser ist dann ein Geheimnis, das nur diesen beiden bekannt ist. Alice wählt sich eine Zufallszahl  $a$  und hält sie geheim. Aus  $a$  berechnet sie mit der Einwegfunktion die Zahl  $A = g^a$  und schickt sie an Bob. Der verfährt ebenso, indem er eine geheime Zufallszahl  $b$  wählt, daraus  $B = g^b$  berechnet und an Alice schickt. Die Zahl  $g$  ist beliebig und darf öffentlich bekannt sein. Alice wendet die Einwegfunktion mit ihrer Geheimzahl  $a$  auf  $B$  an, Bob tut gleiches mit seiner Geheimzahl  $b$  und der empfangenen Zahl  $A$ .

Das Ergebnis  $S$  ist in beiden Fällen dasselbe, weil die Einwegfunktion kommutativ ist:  $g^{a*b} = g^{b*a}$ . Aber selbst Bob kann Alices Geheimnis  $a$  nicht aus den ihm vorliegenden Daten rekonstruieren, Alice wiederum Bobs Geheimnis  $b$  nicht ermitteln, und ein Lauscher, der  $g$  kennt und sowohl  $A$  als auch  $B$  mitgelesen hat, vermag daraus weder  $a$  noch  $b$  noch  $S$  zu berechnen.



<sup>74</sup>In CrypTool ist dieses Austauschprotokoll visualisiert: Sie können die einzelnen Schritte mit konkreten Zahlen nachvollziehen per Menü **Einzelverfahren \ Protokolle \ Diffie-Hellman-Demo**.

<sup>75</sup>Alice und Bob werden standardmäßig als die beiden berechtigten Teilnehmer eines Protokolls bezeichnet (siehe [Schneier1996, Seite 23]).

**Ablauf:**

Alice und Bob wollen also einen geheimen Sessionkey  $S$  über einen abhörbaren Kanal aushandeln.

1. Sie wählen eine Primzahl  $p$  und eine Zufallszahl  $g$ , und tauschen diese Information offen aus.
2. Alice wählt nun  $a$ , eine Zufallszahl kleiner  $p$  und hält diese geheim.  
Bob wählt ebenso  $b$ , eine Zufallszahl kleiner  $p$  und hält diese geheim.
3. Alice berechnet nun  $A \equiv g^a \pmod{p}$ .  
Bob berechnet  $B \equiv g^b \pmod{p}$ .
4. Alice sendet das Ergebnis  $A$  an Bob.  
Bob sendet das Ergebnis  $B$  an Alice.
5. Um den nun gemeinsam zu benutzenden Sessionkey zu bestimmen, potenzieren sie beide jeweils für sich das jeweils empfangene Ergebnis mit ihrer geheimen Zufallszahl modulo  $p$ . Das heißt:
  - Alice berechnet  $S \equiv B^a \pmod{p}$ , und
  - Bob berechnet  $S \equiv A^b \pmod{p}$ .

Auch wenn ein Spion  $g, p$ , und die Zwischenergebnisse  $A$  und  $B$  abhört, kann er den schließlich bestimmten Sessionkey nicht berechnen – wegen der Schwierigkeit, den diskreten Logarithmus<sup>76</sup> zu bestimmen.

Das Ganze soll an einem Beispiel mit (unrealistisch) kleinen Zahlen gezeigt werden.

**Beispiel mit kleinen Zahlen:**

1. Alice und Bob wählen  $g = 11, p = 347$ .
2. Alice wählt  $a = 240$ , Bob wählt  $b = 39$  und behalten  $a$  und  $b$  geheim.
3. Alice berechnet  $A \equiv g^a \equiv 11^{240} \equiv 49 \pmod{347}$ .  
Bob berechnet  $B \equiv g^b \equiv 11^{39} \equiv 285 \pmod{347}$ .
4. Alice sendet Bob:  $A = 49$ ,  
Bob sendet Alice:  $B = 285$ .
5. Alice berechnet  $B^a \equiv 285^{240} \equiv 268 \pmod{347}$ ,  
Bob berechnet  $A^b \equiv 49^{39} \equiv 268 \pmod{347}$ .

Nun können Alice und Bob mit Hilfe ihres gemeinsamen Sessionkeys sicher kommunizieren. Auch wenn ein Spion alles, was über die Leitung ging, abhörte:  $g = 11, p = 347, A = 49$  und  $B = 285$ , den geheimen Schlüssel kann er nicht berechnen.

---

<sup>76</sup>Weitere Details zum Diskreten Logarithmusproblem finden Sie in Kapitel 5.4.

### Bemerkung:

In diesem Beispiel mit den kleinen Zahlen ist das Diskrete Logarithmusproblem leicht lösbar, aber mit großen Zahlen ist es kaum zu lösen<sup>77,78</sup>.

Um die diskreten Logarithmen zu erhalten, ist hier Folgendes zu berechnen:

Von Alice:  $11^x \equiv 49 \pmod{347}$ , also  $\log_{11}(49) \pmod{347}$ .

Von Bob:  $11^y \equiv 285 \pmod{347}$ , also  $\log_{11}(285) \pmod{347}$ .

<sup>77</sup>Mit Sage kann man den diskreten Logarithmus  $x$ , der die Gleichung  $11^x \equiv 49 \pmod{347}$  löst, folgendermaßen bestimmen (hier für Alice): `discrete_log(mod(49, 347), mod(11, 347))`. Als Ergebnis erhält man 67.

Solche zahlentheoretischen Aufgaben können auch mit anderen Tools wie PariGP, LiDIA, BC oder Mathematica (siehe Anhang Web-Links am Ende dieses Kapitel) gelöst werden:

können solche zahlentheoretischen Aufgaben gelöst werden.

- Pari-GP: `znlog(Mod(49,347),Mod(11,347))`.

- LiDIA: `dl(11,49,347)`.

- Mathematica: Die allgemeine Funktion "Solve" liefert die em tdep-Meldung "The equations appear to involve the variables to be solved for in an essentially non-algebraic way".

- Mathematica: `MultiplicativeOrder[11, 347, 49]`.

Alle liefern das Ergebnis 67.

<sup>78</sup>Warum haben die Funktionen für den diskreten Logarithmus für Alice den Wert 67 geliefert und nicht den Wert 240, den Alice als Exponent  $a$  wählte?

Der diskrete Logarithmus ist der kleinste natürliche Exponent, der die Gleichung  $11^x \equiv 49 \pmod{347}$  löst. Sowohl  $x = 67$  als auch  $x = 240$  (die im Beispiel gewählte Zahl) erfüllen die Gleichung und können damit zur Berechnung des Sessionkeys benutzt werden:  $285^{240} \equiv 285^{67} \equiv 268 \pmod{347}$ . Hätten Alice und Bob als Basis  $g$  eine Primitivwurzel modulo  $p$  gewählt, dann gibt es für jeden Rest aus der Menge  $\{1, 2, \dots, p-1\}$  genau einen Exponenten aus der Menge  $\{0, 1, \dots, p-2\}$ .

Info: Zum Modul 347 gibt es 172 verschiedene Primitivwurzeln, davon sind 32 prim (ist nicht notwendig). Da die im Beispiel für  $g$  gewählte Zahl 11 keine Primitivwurzel von 347 ist, nehmen die Reste nicht alle Werte aus der Menge  $\{1, 2, \dots, 346\}$  an. Somit kann es für einen bestimmten Rest mehr als einen oder auch gar keinen Exponenten aus der Menge  $\{0, 1, \dots, 345\}$  geben, der die Gleichung erfüllt.

Mit den entsprechenden Sage-Funktionen findet man:

`is_prime(347)=True, euler_phi(347)=346, gcd(11,347)=1` und `multiplicative_order(mod(11, 347))=173`.

i	$11^i \pmod{347}$	
0	1	
1	11	
2	121	
3	290	
67	49	gesuchter Exponent
172	284	
173	1	= Multiplikative Ordnung von $11^i \pmod{347}$
174	11	
175	121	
176	290	
240	49	gesuchter Exponent

Weitere Details finden Sie in Kapitel 4.18.4 "Primitivwurzeln".

## 4.13 Das RSA-Verfahren mit konkreten Zahlen<sup>79</sup>

Nachdem oben die Funktionsweise des RSA-Verfahrens beschrieben wurde, sollen diese Schritte hier mit konkreten, aber kleinen Zahlen durchgeführt werden.

### 4.13.1 RSA mit kleinen Primzahlen und mit einer Zahl als Nachricht

Bevor wir RSA auf einen Text anwenden, wollen wir es erst direkt mit einer Zahl zeigen<sup>80</sup>.

1. Die gewählten Primzahlen seien  $p = 5$  und  $q = 11$ .  
Also ist  $n = 55$  und  $J(n) = (p - 1) * (q - 1) = 40$ .
2.  $e = 7$  (sollte<sup>81</sup> zwischen 11 und 39 liegen und muss teilerfremd zu 40 sein).
3.  $d = 23$  (da  $23 * 7 \equiv 161 \equiv 1 \pmod{40}$ )  
→ Public-Key des Empfängers:  $(55, 7)$ ,  
→ Private-Key des Empfängers:  $(55, 23)$ .
4. Nachricht sei nur die Zahl  $M = 2$  (also ist kein Aufbrechen in Blöcke nötig).
5. Verschlüsseln:  $C \equiv 2^7 \equiv 18 \pmod{55}$ .
6. Chiffre ist nur die Zahl  $C = 18$  (also kein Aufbrechen in Blöcke nötig).
7. Entschlüsseln:  $M \equiv 18^{23} \equiv 18^{(1+2+4+16)} \equiv 18 * 49 * 36 * 26 \equiv 2 \pmod{55}$ .

Nun wollen wir RSA auf einen Text anwenden: zuerst mit dem Großbuchstabenalphabet (26 Zeichen), dann mit dem gesamten ASCII-Zeichensatz als Bausteine für die Nachrichten.

### 4.13.2 RSA mit etwas größeren Primzahlen und einem Text aus Großbuchstaben

Gegeben ist der Text „ATTACK AT DAWN“ und die Zeichen werden gemäß Tabelle 4.11 codiert<sup>82</sup>.

**Schlüsselerzeugung (Schritt 1 bis 3):**

1.  $p = 47, q = 79$  ( $n = 3713$ ;  $J(n) = (p - 1) * (q - 1) = 3588$ ).
2.  $e = 37$  (sollte<sup>83</sup> zwischen 79 und 3587 liegen und muss teilerfremd zu 3588 sein).
3.  $d = 97$  (denn  $e * d = 1 \pmod{J(n)}$ ;  $37 * 97 \equiv 3589 \equiv 1 \pmod{3588}$ )<sup>84</sup>.

<sup>79</sup>Weiteres Material: Minh Van Nguyen: „Number Theory and the RSA Public Key Cryptosystem. Introductory tutorial on using Sage to study elementary number theory and public key cryptography“, 2009. Didaktisch sehr klarer Artikel zu einigen Grundlagen der Zahlentheorie und zur Benutzung von Sage. [http://nguyenminh2.googlepages.com/sage\\_numtheory-rsa.pdf](http://nguyenminh2.googlepages.com/sage_numtheory-rsa.pdf).

<sup>80</sup>Mit CrypTool können Sie dies per Menü **Einzelverfahren \ RSA-Kryptosystem \ RSA-Demo** lösen.

<sup>81</sup>Siehe Fußnote 47 auf Seite 122.

<sup>82</sup>Mit CrypTool können Sie dies per Menü **Einzelverfahren \ RSA-Kryptosystem \ RSA-Demo** lösen. Dies ist auch im Tutorial/Szenario der Online-Hilfe zu CrypTool beschrieben [Optionen, Alphabet vorgeben, Basissystem, Blocklänge 2 und Dezimaldarstellung].

<sup>83</sup>Siehe Fußnote 47 auf Seite 122.

<sup>84</sup>Wie man  $d = 97$  mit Hilfe des erweiterten ggT berechnet, wird in Anhang 4.14 gezeigt.

Zeichen	Zahlenwert	Zeichen	Zahlenwert
Blank	0	M	13
A	1	N	14
B	2	O	15
C	3	P	16
D	4	Q	17
E	5	R	18
F	6	S	19
G	7	T	20
H	8	U	21
I	9	V	22
J	10	W	23
K	11	X	24
L	12	Y	25
		Z	26

Tabelle 4.11: Großbuchstabenalphabet

#### 4. Verschlüsselung:

Text:    A    T    T    A    C    K            A    T            D    A    W    N  
Zahl:    01   20   20   01   03   11   00   01   20   00   04   01   23   14

Aufteilung dieser 28-stelligen Zahl in 4-stellige Teile (denn 2626 ist noch kleiner als  $n = 3713$ ), d.h. dass die Blocklänge 2 beträgt.

0120 2001 0311 0001 2000 0401 2314

Verschlüsselung aller 7 Teile jeweils per:  $C \equiv M^{37} \pmod{3713}$ <sup>85</sup>:

1404 2932 3536 0001 3284 2280 2235

#### 5. Entschlüsselung:

Chiffre: 1404 2932 3536 0001 3284 2280 2235

Aufteilung dieser 28-stelligen Zahl in 4-stellige Teile.

Entschlüsselung aller 7 Teile jeweils per:  $M \equiv C^{97} \pmod{3713}$ :

0120 2001 0311 0001 2000 0401 2314

Umwandeln von 2-stelligen Zahlen in Großbuchstaben und Blanks.

Bei den gewählten Werten ist es für einen Kryptoanalytiker einfach, aus den öffentlichen Parametern  $n = 3713$  und  $e = 37$  die geheimen Werte zu finden, indem er offenlegt, dass  $3713 = 47 \cdot 79$ .

Wenn  $n$  eine 768-Bit-Zahl ist, bestehen dafür – nach heutigen Kenntnissen – wenig Chancen.

#### 4.13.3 RSA mit noch etwas größeren Primzahlen und mit einem Text aus ASCII-Zeichen

Real wird das ASCII-Alphabet benutzt, um die Einzelzeichen der Nachricht in 8-Bit lange Zahlen zu codieren.

Diese Aufgabe<sup>86</sup> ist angeregt durch das Beispiel aus [Eckert2003, S. 271].

<sup>85</sup>In Kapitel 4.18.5 “RSA-Beispiele mit Sage” finden Sie den Beispiel-Quelltext zur RSA-Verschlüsselung mit Sage. Mit Cryptool können Sie dies per Menü **Einzelverfahren \ RSA-Kryptosystem \ RSA-Demo** lösen.

<sup>86</sup>Mit Cryptool können Sie dies per Menü **Einzelverfahren \ RSA-Kryptosystem \ RSA-Demo** lösen.

Der Text „RSA works!“ bedeutet in Dezimalschreibweise codiert:

Text:	R	S	A		w	o	r	k	s	!
Zahl:	82	83	65	32	119	111	114	107	115	33

Das Beispiel wird in 2 Varianten durchgespielt. Gemeinsam für beide sind die Schritte 1 bis 3.

### Schlüsselerzeugung (Schritt 1 bis 3):

1.  $p = 509$ ,  $q = 503$  ( $n = 256.027$ ;  $J(n) = (p - 1) * (q - 1) = 255.016 = 2^3 * 127 * 251$ )<sup>87</sup>.
2.  $e = 65.537$  (sollte<sup>88</sup> zwischen 509 und 255.015 liegen u. muss<sup>89</sup> teilerfremd zu 255.016 sein).
3.  $d = 231.953$   
(denn  $e \equiv d^{-1} \pmod{J(n)}$ ;  $65.537 * 231.953 \equiv 15.201.503.761 \equiv 1 \pmod{255.016}$ )<sup>90</sup>.

### Variante 1:

Alle ASCII-Zeichen werden einzeln ver- und entschlüsselt (keine Blockbildung).

### 4. Verschlüsselung:

Text:	R	S	A		w	o	r	k	s	!
Zahl:	82	83	65	32	119	111	114	107	115	33

Keine Zusammenfassung der Buchstaben<sup>91</sup>!

Verschlüsselung pro Zeichen per:  $C \equiv M^{65.537} \pmod{256.027}$ <sup>92</sup>:

212984	025546	104529	031692	248407
100412	054196	100184	058179	227433

### 5. Entschlüsselung:

Chifftrat:

212984	025546	104529	031692	248407
100412	054196	100184	058179	227433

Entschlüsselung pro Zeichen per:  $M \equiv C^{231.953} \pmod{256.027}$ :

82 83 65 32 119 111 114 107 115 33

<sup>87</sup>In Kapitel 4.18.5 „RSA-Beispiele mit Sage“ finden Sie den Quelltext zur Faktorisierung von  $J(n)$  mit Sage.

Mit CrypTool können Sie dies per Menü **Einzelverfahren \ RSA-Kryptosystem \ Faktorisieren einer Zahl** lösen.

<sup>88</sup>Siehe Fußnote 47 auf Seite 122.

<sup>89</sup> $e$  darf also nicht 2, 127 oder 251 sein ( $65537 = 2^{16} + 1$ ) ( $255,016 = 2^3 * 127 * 251$ ).

Real wird  $J(n)$  nicht faktorisiert, sondern für das gewählte  $e$  wird mit dem Euklidischen Algorithmus sichergestellt, dass  $\text{ggT}(e, J(n)) = 1$ .

<sup>90</sup>Andere mögliche Kombinationen von  $(e, d)$  sind z.B.: (3, 170.011), (5, 204.013), (7, 36.431).

<sup>91</sup>Für sichere Verfahren braucht man große Zahlen, die möglichst alle Werte bis  $n - 1$  annehmen. Wenn die mögliche Wertemenge der Zahlen in der Nachricht zu klein ist, nutzen auch große Primzahlen nichts für die Sicherheit. Ein ASCII-Zeichen ist durch 8 Bits repräsentiert. Will man größere Werte, muss man mehrere Zeichen zusammenfassen. Zwei Zeichen benötigen 16 Bit, womit maximal der Wert 65.536 darstellbar ist; dann muss der Modul  $n$  größer sein als  $2^{16} = 65.536$ . Dies wird in Variante 2 angewandt. Beim Zusammenfassen bleiben in der Binärschreibweise die führenden Nullen erhalten (genauso wie wenn man oben in der Dezimalschreibweise alle Zahlen 3-stellig schreiben würde und dann die Folge 082 083, 065 032, 119 111, 114 107, 115 033 hätte).

<sup>92</sup>In Kapitel 4.18.5 „RSA-Beispiele mit Sage“ finden Sie den Quelltext zur RSA-Exponentiation mit Sage.

## Variante 2:

Jeweils zwei ASCII-Zeichen werden als Block ver- und entschlüsselt.

Bei der Variante 2 wird die Blockbildung in zwei verschiedenen Untervarianten 4./5. und 4'./5'. dargestellt.

Text:	R	S	A		w	o	r	k	s	!
Zahl:	82	83	65	32	119	111	114	107	115	33

### 4. Verschlüsselung:

Blockbildung<sup>93</sup> (die ASCII-Zeichen werden als 8-stellige Binärzahlen hintereinander geschrieben):

21075 16672 30575 29291 29473<sup>94</sup>

Verschlüsselung pro Block per:  $C \equiv M^{65.537} \pmod{256027}$ <sup>95</sup>:

158721 137346 37358 240130 112898

### 5. Entschlüsselung:

Chifftrat:

158721 137346 37358 240130 112898

Entschlüsselung pro Block per:  $M \equiv C^{231.953} \pmod{256.027}$ :

21075 16672 30575 29291 29473

### 4'. Verschlüsselung:

Blockbildung (die ASCII-Zeichen werden als 3-stellige Dezimalzahlen hintereinander geschrieben):

82083 65032 119111 114107 115033<sup>96</sup>

Verschlüsselung pro Block per:  $C \equiv M^{65537} \pmod{256.027}$ <sup>97</sup>:

198967 051405 254571 115318 014251

### 5'. Entschlüsselung:

Chifftrat:

198967 051405 254571 115318 014251

Entschlüsselung pro Block per:  $M \equiv C^{231.953} \pmod{256.027}$ :

82083 65032 119111 114107 115033

<sup>93</sup>

Einzelzeichen	Binärdarstellung	Dezimaldarstellung
01010010, 82	01010010 01010011	= 21075
01010011, 83		
01000001, 65	01000001 00100000	= 16672
00100000, 32		
01110111, 119	01110111 01101111	= 30575
01101111, 111		
01110010, 114	01110010 01101011	= 29291
01101011, 107		
01110011, 115	01110011 00100001	= 29473
00100001, 33:		

<sup>94</sup>Mit CrypTool können Sie dies per Menü **Einzelverfahren \ RSA-Kryptosystem \ RSA-Demo** mit den folgenden Optionen lösen: alle 256 Zeichen, b-adisch, Blocklänge 2, dezimale Darstellung.

<sup>95</sup>In Kapitel 4.18.5 "RSA-Beispiele mit Sage" finden Sie den Quelltext zur RSA-Exponentiation mit Sage.

<sup>96</sup>Die RSA-Verschlüsselung mit dem Modul  $n = 256.027$  ist bei dieser Einstellung korrekt, da die ASCII-Blöcke in Zahlen kleiner oder gleich 255.255 kodiert werden.

<sup>97</sup>In Kapitel 4.18.5 "RSA-Beispiele mit Sage" finden Sie den Quelltext zur RSA-Exponentiation mit Sage.

#### 4.13.4 Eine kleine RSA-Cipher-Challenge (1)

Die Aufgabe stammt aus [Stinson1995, Exercise 4.6]: Die pure Lösung hat Prof. Stinson unter <http://www.cacr.math.uwaterloo.ca/~dstinson/solns.html><sup>98</sup>

veröffentlicht. Es geht aber nicht nur um das Ergebnis, sondern vor allem um die Einzelschritte der Lösung, also um die Darlegung der Kryptoanalyse<sup>99</sup>.

Hier die Aufgabe im Originaltext:

Two samples of RSA ciphertext are presented in Tables 4.12<sup>100</sup> and 4.13<sup>101</sup>. Your task is to decrypt them. The public parameters of the system are

$n = 18.923$  and  $e = 1261$  (for Table 4.12) and

$n = 31.313$  and  $e = 4913$  (for Table 4.13).

This can be accomplished as follows. First, factor  $n$  (which is easy because it is so small). Then compute the exponent  $d$  from  $J(n)$ , and, finally, decrypt the ciphertext. Use the square-and-multiply algorithm to exponentiate modulo  $n$ .

In order to translate the plaintext back into ordinary English text, you need to know how alphabetic characters are "encoded" as elements in  $\mathbb{Z}_n$ . Each element of  $\mathbb{Z}_n$  represents three alphabetic characters as in the following examples:

$$\text{DOG} \mapsto 3 * 26^2 + 14 * 26 + 6 = 2398$$

$$\text{CAT} \mapsto 2 * 26^2 + 0 * 26 + 19 = 1371$$

$$\text{ZZZ} \mapsto 25 * 26^2 + 25 * 26 + 25 = 17.575.$$

You will have to invert this process as the final step in your program.

The first plaintext was taken from "The Diary of Samuel Marchbanks", by Robertson Davies, 1947, and the second was taken from "Lake Wobegon Days", by Garrison Keillor, 1985.

#### 4.13.5 Eine kleine RSA-Cipher-Challenge (2)

Die folgende Aufgabe ist eine korrigierte Variante aus dem Buch von Prof. Yan [Yan2000, Example 3.3.7, S. 318]. Es geht aber nicht nur um das Ergebnis, sondern vor allem um die Einzelschritte der Lösung, also um die Darlegung der Kryptoanalyse<sup>102</sup>.

Man kann sich drei völlig unterschiedlich schwierige Aufgaben vorstellen: Gegeben ist jeweils der Geheimtext und der öffentliche Schlüssel  $(e, n)$ :

- Known-Plaintext: finde den geheimen Schlüssel  $d$  unter Benutzung der zusätzlich bekannten Ursprungsnachricht.
- Ciphertext-only: finde  $d$  und die Klartextnachricht.
- RSA-Modul knacken, d.h. faktorisieren (ohne Kenntnis der Nachrichten).

<sup>98</sup>oder <http://bibd.unl/~stinson/solns.html>.

<sup>99</sup>Im Szenario der Online-Hilfe zu CrypTool und in der Präsentation auf der Web-Seite wird der Lösungsweg skizziert. Wenn uns jemand einen (weiteren) gut aufbereiteten konkreten Lösungsweg schickt, nehmen wir ihn gerne in die Dokumentation auf.

<sup>100</sup>Die Zahlen dieser Tabelle können Sie mit Copy und Paste weiter bearbeiten.

<sup>101</sup>Die Zahlen dieser Tabelle befinden sich auch in der Online-Hilfe „Szenario für die RSA-Demonstration“ der CrypTool-Auslieferung.

<sup>102</sup>Im Szenario der Online-Hilfe zu CrypTool und in der CrypTool-Präsentation werden der Lösungsweg skizziert. Wenn uns jemand einen gut aufbereiteten konkreten Lösungsweg schickt, nehmen wir ihn gerne in die Dokumentation auf.



12423	11524	7243	7459	14303	6127	10964	16399
9792	13629	14407	18817	18830	13556	3159	16647
5300	13951	81	8986	8007	13167	10022	17213
2264	961	17459	4101	2999	14569	17183	15827
12693	9553	18194	3830	2664	13998	12501	18873
12161	13071	16900	7233	8270	17086	9792	14266
13236	5300	13951	8850	12129	6091	18110	3332
15061	12347	7817	7946	11675	13924	13892	18031
2620	6276	8500	201	8850	11178	16477	10161
3533	13842	7537	12259	18110	44	2364	15570
3460	9886	8687	4481	11231	7547	11383	17910
12867	13203	5102	4742	5053	15407	2976	9330
12192	56	2471	15334	841	13995	17592	13297
2430	9741	11675	424	6686	738	13874	8168
7913	6246	14301	1144	9056	15967	7328	13203
796	195	9872	16979	15404	14130	9105	2001
9792	14251	1498	11296	1105	4502	16979	1105
56	4118	11302	5988	3363	15827	6928	4191
4277	10617	874	13211	11821	3090	18110	44
2364	15570	3460	9886	9988	3798	1158	9872
16979	15404	6127	9872	3652	14838	7437	2540
1367	2512	14407	5053	1521	297	10935	17137
2186	9433	13293	7555	13618	13000	6490	5310
18676	4782	11374	446	4165	11634	3846	14611
2364	6789	11634	4493	4063	4576	17955	7965
11748	14616	11453	17666	925	56	4118	18031
9522	14838	7437	3880	11476	8305	5102	2999
18628	14326	9175	9061	650	18110	8720	15404
2951	722	15334	841	15610	2443	11056	2186

Tabelle 4.12: RSA-Geheimtext A

$n = 63978486879527143858831415041$ ,  $e = 17579$

Klartextnachricht<sup>103</sup>:

1401202118011200,  
1421130205181900,  
0118050013010405,  
0002250007150400

Geheimtext:

45411667895024938209259253423,  
16597091621432020076311552201,  
46468979279750354732637631044,  
32870167545903741339819671379

### Bemerkung:

Die ursprüngliche Nachricht bestand aus einem Satz mit 31 Zeichen (codiert mit dem Großbuch-

<sup>103</sup>Die Zahlen dieser Tabelle befinden sich auch in der Online-Hilfe „Szenario für die RSA-Demonstration“ der CrypTool-Auslieferung.

6340	8309	14010	8936	27358	25023	16481	25809
23614	7135	24996	30590	27570	26486	30388	9395
27584	14999	4517	12146	29421	26439	1606	17881
25774	7647	23901	7372	25774	18436	12056	13547
7908	8635	2149	1908	22076	7372	8686	1304
4082	11803	5314	107	7359	22470	7372	22827
15698	30317	4685	14696	30388	8671	29956	15705
1417	26905	25809	28347	26277	7897	20240	21519
12437	1108	27106	18743	24144	10685	25234	30155
23005	8267	9917	7994	9694	2149	10042	27705
15930	29748	8635	23645	11738	24591	20240	27212
27486	9741	2149	29329	2149	5501	14015	30155
18154	22319	27705	20321	23254	13624	3249	5443
2149	16975	16087	14600	27705	19386	7325	26277
19554	23614	7553	4734	8091	23973	14015	107
3183	17347	25234	4595	21498	6360	19837	8463
6000	31280	29413	2066	369	23204	8425	7792
25973	4477	30989					

Tabelle 4.13: RSA-Geheimtext B

stabenalphabet aus Abschnitt 4.13.2). ann wurden je 16 Dezimalziffern zu einer Zahl zusammengefasst (die letzte Zahl wurde mit Nullen aufgefüllt). Diese Zahlen wurden mit  $e$  potenziert.

Beim Entschlüsseln ist darauf zu achten, dass die berechneten Zahlen vorne mit Nullen aufzufüllen sind, um den Klartext zu erhalten.

Wir betonen das, weil in der Implementierung und Standardisierung die Art des Padding sehr wichtig ist für interoperable Algorithmen.

## 4.14 Anhang: Der größte gemeinsame Teiler (ggT) von ganzen Zahlen und die beiden Algorithmen von Euklid<sup>104</sup>

Der größte gemeinsame Teiler zweier natürlicher Zahlen  $a$  und  $b$  ist eine wichtige Größe, die sehr schnell berechnet werden kann. Wenn eine Zahl  $c$  die Zahlen  $a$  und  $b$  teilt (d.h. es gibt ein  $a'$  und ein  $b'$ , so dass  $a = a' * c$  und  $b = b' * c$ ), dann teilt  $c$  auch den Rest  $r$  der Division von  $a$  durch  $b$ . Wir schreiben in aller Kürze: Aus  $c$  teilt  $a$  und  $b$  folgt:  $c$  teilt  $r = a - \lfloor a/b \rfloor * b$ <sup>105</sup>.

Weil die obige Aussage für alle gemeinsamen Teiler  $c$  von  $a$  und  $b$  gilt, folgt für den größten gemeinsamen Teiler von  $a$  und  $b$  ( $\text{ggT}(a, b)$ ) die Aussage

$$\text{ggT}(a, b) = \text{ggT}(a - \lfloor a/b \rfloor * b, b).$$

Mit dieser Erkenntnis lässt sich der Algorithmus zum Berechnen des ggT zweier Zahlen wie folgt (in Pseudocode) beschreiben:

```
INPUT: a, b != 0
1. if ( a < b ) then  x = a; a = b; b = x; // Vertausche a und b (a > b)
2. a = a - int(a/b) * b          // a wird kleiner b, der ggT(a, b)
                                // bleibt unverändert
3. if ( a != 0 ) then goto 1.   // nach jedem Schritt fällt a, und
                                // der Algorithmus endet, wenn a == 0.
OUTPUT "ggT(a,b) = " b         // b ist der ggT vom ursprünglichen a und b
```

Aus dem ggT lassen sich aber noch weitere Zusammenhänge bestimmen: Dazu betrachtet man für  $a$  und  $b$  das Gleichungssystem:

$$\begin{aligned} a &= 1 * a + 0 * b \\ b &= 0 * a + 1 * b, \end{aligned}$$

bzw. in Matrix-Schreibweise:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} a \\ b \end{pmatrix}.$$

Wir fassen diese Informationen in der erweiterten Matrix

$$\left( \begin{array}{cc|cc} a & & 1 & 0 \\ b & & 0 & 1 \end{array} \right)$$

zusammen. Wendet man den ggT-Algorithmus auf diese Matrix an, so erhält man den *erweiterten Euklidschen Algorithmus*, mit dem die multiplikative Inverse bestimmt wird.

<sup>104</sup>Mit dem Zahlentheorie-Lernprogramm **ZT** können Sie sehen,

a) wie Euklids Algorithmus den ggT berechnet (Lern-Kapitel 1.3, Seiten 14-19/21) und

b) wie Euklids erweiterter Algorithmus das multiplikative Inverse findet (Lern-Kapitel 2.2, Seite 13/40).

ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

<sup>105</sup>Die Gaussklammer  $\lfloor x \rfloor$  der reellwertigen Zahl  $x$  ist definiert als:  $\lfloor x \rfloor$  ist die größte ganze Zahl kleiner oder gleich  $x$ .

INPUT:  $a, b \neq 0$

0.  $x_{1,1} := 1, x_{1,2} := 0, x_{2,1} := 0, x_{2,2} := 1$

1.  $\left( \begin{array}{c|cc} a & x_{1,1} & x_{1,2} \\ b & x_{2,1} & x_{2,2} \end{array} \right) := \left( \begin{array}{cc} 0 & 1 \\ 1 & -\lfloor a/b \rfloor * b \end{array} \right) * \left( \begin{array}{c|cc} a & x_{1,1} & x_{1,2} \\ b & x_{2,1} & x_{2,2} \end{array} \right).$

2. if (b != 0) then goto 1.

OUTPUT: "ggT( $a, b$ ) =  $a * x + b * y$ :", "ggT( $a, b$ ) = "  $b$ , " $x =$ "  $x_{2,1}$ , " $y =$ "  $x_{2,2}$

Da dieser Algorithmus nur lineare Transformationen durchführt, gelten immer die Gleichungen

$$\begin{aligned} a &= x_{1,1} * a + x_{1,2} * b \\ b &= x_{2,1} * a + x_{2,2} * b, \end{aligned}$$

Am Ende liefert der Algorithmus<sup>106</sup> die erweiterte ggT-Gleichung:

$$\gcd(a, b) = a * x_{2,1} + b * x_{2,2}.$$

### Beispiel:

Mit dem erweiterten ggT lässt sich für  $e = 37$  die modulo 3588 multiplikativ inverse Zahl  $d$  bestimmen (d.h.  $37 * d \equiv 1 \pmod{3588}$ ):

$$0. \left( \begin{array}{c|cc} 3588 & 1 & 0 \\ 37 & 0 & 1 \end{array} \right)$$

$$1. \left( \begin{array}{c|cc} 37 & 1 & 0 \\ 36 & 0 & -96 \end{array} \right) = \left( \begin{array}{cc} 0 & 1 \\ 1 & -(\lfloor 3588/36 \rfloor = 96) * 37 \end{array} \right) * \left( \begin{array}{c|cc} 3588 & 1 & 0 \\ 37 & 0 & 1 \end{array} \right).$$

$$2. \left( \begin{array}{c|cc} 36 & 1 & -96 \\ 1 & -1 & 97 \end{array} \right) = \left( \begin{array}{cc} 0 & 1 \\ 1 & -(\lfloor 37/36 \rfloor = 1) * 36 \end{array} \right) * \left( \begin{array}{c|cc} 37 & 1 & 0 \\ 36 & 0 & -96 \end{array} \right).$$

$$3. \left( \begin{array}{c|cc} 1 & -1 & 97 \\ 0 & 37 & -3588 \end{array} \right) = \left( \begin{array}{cc} 0 & 1 \\ 1 & -(\lfloor 36/1 \rfloor = 36) * 1 \end{array} \right) * \left( \begin{array}{c|cc} 36 & 1 & -96 \\ 1 & -1 & 97 \end{array} \right).$$

OUTPUT:

ggT(37, 3588) =  $a * x + b * y$ :

ggT(37, 3588) = 1,  $x = -1$ ,  $y = 97$ .

Es folgt

1. 37 und 3588 sind teilerfremd (37 ist invertierbar modulo 3588).

2.  $37 * 97 = (1 * 3588) + 1$  mit anderen Worten  $37 * 97 \equiv 1 \pmod{3588}$   
und damit ist die Zahl 97 multiplikativ invers zu 37 modulo 3588.

<sup>106</sup>Wenn der ggT-Algorithmus endet, steht in den Programmvariablen  $a$  und  $b$ :  $a = 0$  und  $b = \gcd(a, b)$ . Bitte beachten Sie, dass die Programmvariablen zu den Zahlen  $a$  und  $b$  verschieden sind und ihre Gültigkeit nur im Rahmen des Algorithmus haben.

## 4.15 Anhang: Abschlussbildung

Die Eigenschaft der Abgeschlossenheit innerhalb einer Menge wird immer bezüglich einer Operation definiert. Im folgenden wird gezeigt, wie man für eine gegebene Ausgangsmenge  $G_0$  die abgeschlossene Menge  $G$  bezüglich der Operation  $+$  (mod 8) konstruiert:

$G_0 = \{2, 3\}$  — — — die Addition der Zahlen in  $G_0$  bestimmt weitere Zahlen :

$$2 + 3 \equiv 5 \pmod{8} = 5$$

$$2 + 2 \equiv 4 \pmod{8} = 4$$

$$3 + 3 \equiv 6 \pmod{8} = 6$$

$G_1 = \{2, 3, 4, 5, 6\}$  — — — die Addition der Zahlen in  $G_1$  bestimmt :

$$3 + 4 \equiv 7 \pmod{8} = 7$$

$$3 + 5 \equiv 8 \pmod{8} = 0$$

$$3 + 6 \equiv 9 \pmod{8} = 1$$

$G_2 = \{0, 1, 2, 3, 4, 5, 6, 7\}$  — — — die Addition der Zahlen in  $G_2$  *erweitert die Menge nicht!*

$G_3 = G_2$  — — — man sagt :  $G_2$  ist abgeschlossen bezüglich der Addition (mod 8).

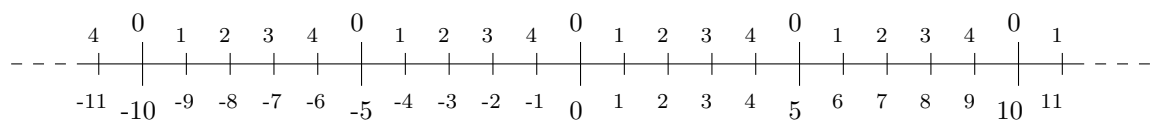
## 4.16 Anhang: Bemerkungen zur modulo Subtraktion

Beispielweise gilt für die Subtraktion modulo 5:  $2 - 4 = -2 \equiv 3 \pmod{5}$ .

Es gilt also nicht, dass:  $-2 = 2 \pmod{5}$  !

Dies gleichzusetzen ist ein häufig gemachter Fehler. Warum diese nicht gleich ist, kann man sich gut verdeutlichen, wenn man die Permutation  $(0, 1, 2, 3, 4)$  aus  $\mathbb{Z}_5$  von z.B.  $-11$  bis  $+11$  wiederholt über den Zahlenstrahl aus  $\mathbb{Z}$  legt.

Zahlengerade modulo 5



Zahlengerade der ganzen Zahlen

## 4.17 Anhang: Basisdarstellung und -umwandlung von Zahlen, Abschätzung der Ziffernlänge

Betrachtet man eine Zahl  $z$ , so stellt sich die Frage, wie man sie darstellt. Üblich sind die Schreibweisen  $z = 2374$  oder  $z = \sqrt{2}$ . Die zweite Zahl kann nicht in der ersten Form dargestellt werden, da sie unendlich viele Stellen hat. Man umgeht das Problem durch die symbolische Schreibweise. Muss man solche Zahlen in der Zifferschreibweise darstellen, bleibt nichts anderes übrig als die Zahl zu runden.

Wir haben uns an die Dezimalschreibweise (Zahlen zur Basis 10) gewöhnt. Computer rechnen intern mit Zahlen im Binärformat, die nur bei der Ausgabe in Dezimalschreibweise oder auch manchmal in Hexadezimalschreibweise (Basis 16) dargestellt werden.

Dieser Anhang beschreibt, wie man die Basisdarstellung einer natürlichen Zahl umrechnet in die Darstellung derselben Zahl mit einer anderen Basis, und wie man mit Hilfe der Logarithmusfunktion die Ziffernlänge jeder natürlichen Zahl zu einer beliebigen Basis abschätzen kann.

### $b$ -adische Summendarstellung von natürlichen Zahlen

Zur Basis  $b$  kann man jede natürliche Zahl  $z$  darstellen als eine  $b$ -adische Summe von Zahlen.

$$z = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b + a_0,$$

wobei die natürlichen Zahlen  $a_i$ ,  $i = 0, \dots, n$  aus dem Wertebereich  $0, 1, 2, \dots, b-1$  gewählt sind. Wir nennen diese Zahlen  $a_i$  Ziffern.

Für diese spezielle Summe gilt:

- 1) Für beliebige Ziffern  $a_0, a_1, \dots, a_n$  gilt:  $b^{n+1} > a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b + a_0$ .
  - 2) Es gibt auch Ziffern  $a_0, a_1, \dots, a_n$  (nämlich  $a_i = b-1$  für  $i = 0, \dots, n$ ), so dass  $b^{n+1} - 1 \leq a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b + a_0$ .
- (Damit lässt sich leicht zeigen, dass sich jede natürliche Zahl als  $b$ -adische Summe darstellen lässt).

Wenn man diese Ziffern  $a_n a_{n-1} \cdots a_1 a_0$  direkt hintereinander schreibt und die Gewichte  $b^i$  weglässt, so erhält man die gewohnte Schreibweise für Zahlen.

#### Beispiel:

Basis  $b = 10$ :  $10278 = 1 \cdot 10^4 + 0 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 8$

Basis  $b = 16$ :  $FE70A = 15 \cdot 16^4 + 14 \cdot 16^3 + 7 \cdot 16^2 + 0 \cdot 16^1 + 10$ .

### Länge der Zifferndarstellung

Für eine natürliche Zahl  $z$  kann die Länge der Zahl in  $b$ -adischer Darstellung wie folgt bestimmt werden. Dazu geht man von der Abschätzung  $b^{n+1} > z \geq b^n$  aus, wobei  $n+1$  die gesuchte Stellenzahl ist. Nach dem Logarithmieren zur Basis  $b$ <sup>107</sup> bestimmt man die Ungleichung  $n+1 > \log_b z \geq n$ . Es folgt, dass  $n = \lfloor \log_b z \rfloor$ <sup>108</sup>. Man bezeichnet mit  $l_b(z)$  die Zahl der benötigten

<sup>107</sup>Nach den Logarithmengesetzen gilt für die Basen  $b$  und  $b'$  die Gleichung  $\log_b z = \log_{b'} z / \log_{b'}(b)$ . Es ist also einfach, ausgehend von Logarithmentafeln für z.B. die Basis  $b' = 10$  den Logarithmus zur Basis  $b = 2$  zu berechnen.

<sup>108</sup>Die Funktion  $\lfloor x \rfloor$  bestimmt die nächste ganze Zahl kleiner  $x$  (wenn  $x \geq 0$ , dann werden die Nachkommastellen der Zahl einfach abgeschnitten). Man sagt auch *untere Gaußklammer von  $x$* .

Ziffern zur Darstellung der Zahl  $z$  in  $b$ -adischer Schreibweise. Es gilt die Gleichung

$$l_b(z) = \lfloor \log_b z \rfloor + 1.$$

**Beispiel 1 (dezimal→hex):**

Für die Zahl  $z = 234$  in Dezimalschreibweise (EA in hex) bestimmt man die Länge in Hexadezimalschreibweise durch

$$l_{16}(z) = \lfloor \log_{16}(z) \rfloor + 1 = \lfloor \ln(z)/\ln(16) \rfloor + 1 = \lfloor 1,96\dots \rfloor + 1 = 1 + 1 = 2.$$

**Beispiel 2 (dezimal→binär):**

Für die Zahl  $z = 234$  in Dezimalschreibweise (11101010 in binär) bestimmt man die Länge in Binärschreibweise durch

$$l_2(z) = \lfloor \log_2(z) \rfloor + 1 = \lfloor \ln(z)/\ln(2) \rfloor + 1 = \lfloor 7,87\dots \rfloor + 1 = 7 + 1 = 8.$$

**Beispiel 3 (binär→dezimal):**

Für die Zahl  $z = 11101010$  in Binärschreibweise (234 dezimal) bestimmt man die Länge in Dezimalschreibweise durch

$$l_{10}(z) = \lfloor \log_{10}(z) \rfloor + 1 = \lfloor \ln(z)/\ln(10) \rfloor + 1 = \lfloor 2,36\dots \rfloor + 1 = 2 + 1 = 3.$$

## Algorithmus zur Basisdarstellung von Zahlen

Ausgehend von der Zahl  $z$  bestimmt man die Zahlendarstellung zur Basis  $b$  durch den folgenden Algorithmus:

**input:**  $z, b$

$n := 0, z' := z$

**while**  $z' > 0$  **do**

$a_n := z' \bmod b,$

$z' := \lfloor z'/b \rfloor$

$n := n + 1$

**end do**

**output:**  $a_n a_{n-1} \dots a_1 a_0$  Zahlendarstellung zur Basis  $b$ .

**Beispiel 1 (dezimal→hex):**

Die Zahl  $z = 234$  in Dezimalschreibweise wird umgewandelt in Hexadezimalschreibweise:

$a_0 = 234 \bmod 16 = 10 = A, 234/16 = 14 = E,$

$a_1 = 14 \bmod 16 = E$

Damit ergibt sich EA.

**Beispiel 2 (binär→dezimal):**

Die Binärzahl  $z = 1000100101110101$  wird in die Dezimaldarstellung durch die folgenden Berechnungen umgewandelt:

$1000100101110101 = 1001 \pmod{1010} \implies a_0 = 9, \quad 1000100101110101/1010 = 110110111110$   
 $110110111110 = 1000 \pmod{1010} \implies a_1 = 8, \quad 110110111110/1010 = 101011111$   
 $101011111 = 1 \pmod{1010} \implies a_2 = 1, \quad 10101111/1010 = 100011$   
 $100011 = 101 \pmod{1010} \implies a_3 = 5, \quad 100011/1010 = 1$   
 $11 = 11 \pmod{1010} \implies a_4 = 3$   
 Damit ergibt sich  $z = 35189$ .



## 4.18 Anhang: Beispiele mit Sage

In diesem Anhang finden Sie Sage-Quellcode, mit dem Sie die Tabellen und Beispiele des Kapitels 4 (“Einführung in die elementare Zahlentheorie mit Beispielen”) berechnen können.

### 4.18.1 Multiplikationstabellen modulo $m$

Die Multiplikationstabelle 4.4 (von Seite 108) für  $a \times i \pmod{m}$ , mit  $m = 17$ ,  $a = 5$  und  $a = 6$ , und  $i$  von 0 bis 16 kann mit folgenden Sage-Befehlen berechnet werden:

---

**Sage-Beispiel 4.1** Multiplikationstabelle  $a \times i \pmod{m}$  mit  $m = 17$ ,  $a = 5$  und  $a = 6$

---

```
sage: m = 17; a = 5; b = 6
sage: [mod(a * i, m).lift() for i in xrange(m)]
[0, 5, 10, 15, 3, 8, 13, 1, 6, 11, 16, 4, 9, 14, 2, 7, 12]
sage: [mod(b * i, m).lift() for i in xrange(m)]
[0, 6, 12, 1, 7, 13, 2, 8, 14, 3, 9, 15, 4, 10, 16, 5, 11]
```

---

Die Funktion `mod()` gibt das Objekt zurück, das die natürlichen Zahlen modulo  $m$  (in unserem Fall  $m = 17$ ) repräsentiert. Aus dem `Mod`-Objekt kann man die einzelnen Komponenten entweder mit der `component-` oder mit der `lift`-Funktion zurückgewinnen. Wir nutzen hier die Methode `lift()`, um das Objekt in eine Zahl umzuwandeln und auszugeben.

Die weiteren Beispiele der Multiplikationstabelle modulo 13 (table 4.5) und modulo 12 (table 4.6) auf Seite 108 kann man auf dieselbe Weise bestimmen, wenn man im Quelltext jeweils `m=17` durch den entsprechenden Zahlenwert (`m=13` bzw. `m=12`) ersetzt.

### 4.18.2 Schnelles Berechnen hoher Potenzen

Das schnelle Potenzieren modulo  $m$  kann mit der Sage-Funktion `power_mod()` durchgeführt werden. Das Ergebnis dieser Funktion ist eine natürliche Zahl. Sie können mit Hilfe der folgenden Zeilen die Idee der Square-and-Multiply-Methode nachvollziehen, wie sie im Beispiel in Kapitel “Schnelles Berechnen hoher Potenzen” auf Seite 111 dargestellt ist:

---

**Sage-Beispiel 4.2** Schnelles Berechnen hoher Potenzen mod  $m = 103$

---

```
sage: a = 87; m = 103
sage: exp = [2, 4, 8, 16, 32, 43]
sage: [power_mod(a, e, m) for e in exp]
[50, 28, 63, 55, 38, 85]
```

---

### 4.18.3 Multiplikative Ordnung

Die Ordnung  $\text{ord}_m(a)$  einer Zahl  $a$  in der multiplikativen Gruppe  $Z_m^*$  ist die kleinste natürliche Zahl  $i \geq 1$  für die gilt  $a^i \equiv 1 \pmod{m}$  (siehe Kapitel 4.9, „Multiplikative Ordnung und Primivwurzel“).

Um die Tabelle 4.7 auf Seite 119 zu berechnen, können wir alle Potenzen  $a^i \pmod{11}$  wie folgt ausgeben:

---

**Sage-Beispiel 4.3** Tabelle mit allen Potenzen  $a^i \pmod{m}$  für  $m = 11$ ,  $a = 1, \dots, 10$

---

```
sage: m = 11
sage: for a in xrange(1, m):
....:     print [power_mod(a, i, m) for i in xrange(1, m)]
....:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[2, 4, 8, 5, 10, 9, 7, 3, 6, 1]
[3, 9, 5, 4, 1, 3, 9, 5, 4, 1]
[4, 5, 9, 3, 1, 4, 5, 9, 3, 1]
[5, 3, 4, 9, 1, 5, 3, 4, 9, 1]
[6, 3, 7, 9, 10, 5, 8, 4, 2, 1]
[7, 5, 2, 3, 10, 4, 6, 9, 8, 1]
[8, 9, 6, 4, 10, 3, 2, 5, 7, 1]
[9, 4, 3, 5, 1, 9, 4, 3, 5, 1]
[10, 1, 10, 1, 10, 1, 10, 1, 10, 1]
```

und die letzte Spalte ergänzt um die Ordnung des jeweiligen  $a \pmod{11}$

```
sage: m = 11
sage: for a in xrange(1, m):
....:     lst= [power_mod(a, i, m) for i in xrange(1, m)]
....:     lst.append(multiplicative_order(mod(a,m)))
....:     print lst
....:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[2, 4, 8, 5, 10, 9, 7, 3, 6, 1, 10]
[3, 9, 5, 4, 1, 3, 9, 5, 4, 1, 5]
[4, 5, 9, 3, 1, 4, 5, 9, 3, 1, 5]
[5, 3, 4, 9, 1, 5, 3, 4, 9, 1, 5]
[6, 3, 7, 9, 10, 5, 8, 4, 2, 1, 10]
[7, 5, 2, 3, 10, 4, 6, 9, 8, 1, 10]
[8, 9, 6, 4, 10, 3, 2, 5, 7, 1, 10]
[9, 4, 3, 5, 1, 9, 4, 3, 5, 1, 5]
[10, 1, 10, 1, 10, 1, 10, 1, 10, 1, 2]
```

---

Die Tabelle 4.8 auf Seite 119 enthält Beispiele für die Ordnung von  $a$  modulo 45 ( $\text{ord}_{45}(a)$ ) und den Wert der Eulerfunktion von 45 ( $J(45)$ ).

Der folgende Sage-Code erzeugt eine analoge Tabelle.

---

**Sage-Beispiel 4.4** Tabelle mit allen Potenzen  $a^i \pmod{45}$  für  $a = 1, \dots, 12$  plus der Ordnung von  $a$

---

```
sage: m = 45
sage: for a in xrange(1, 13):
....:     lst = [power_mod(a, i, m) for i in xrange(1, 13)]
....:     try:
....:         lst.append(multiplicative_order(mod(a, m)))
....:     except:
....:         lst.append("None")
....:     lst.append(euler_phi(m))
....:     print lst
....:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 24]
[2, 4, 8, 16, 32, 19, 38, 31, 17, 34, 23, 1, 12, 24]
[3, 9, 27, 36, 18, 9, 27, 36, 18, 9, 27, 36, 'None', 24]
[4, 16, 19, 31, 34, 1, 4, 16, 19, 31, 34, 1, 6, 24]
[5, 25, 35, 40, 20, 10, 5, 25, 35, 40, 20, 10, 'None', 24]
[6, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 'None', 24]
[7, 4, 28, 16, 22, 19, 43, 31, 37, 34, 13, 1, 12, 24]
[8, 19, 17, 1, 8, 19, 17, 1, 8, 19, 17, 1, 4, 24]
[9, 36, 9, 36, 9, 36, 9, 36, 9, 36, 9, 36, 'None', 24]
[10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 'None', 24]
[11, 31, 26, 16, 41, 1, 11, 31, 26, 16, 41, 1, 6, 24]
[12, 9, 18, 36, 27, 9, 18, 36, 27, 9, 18, 36, 'None', 24]
```

---

Die Ordnung  $\text{ord}_m(a)$  kann nur berechnet werden, wenn  $a$  teilerfremd zu  $m$  ist. Das kann mit der Abfrage, ob  $\text{gcd}(a, m) == 1$ , überprüft werden.

In unserem Codebeispiel haben wir stattdessen die Berechnung der multiplikativen Ordnung in einem **try-except**-Block durchgeführt. Auf diese Weise kann Sage jede Ausnahme und jeden Fehler abfangen, der von der Funktion `multiplicative_order()` geworfen wird. Wird eine Ausnahmen oder ein Fehler im **try**-Block geworfen, wissen wir, dass  $\text{ord}_m(a)$  nicht existiert für den gegebenen Wert von  $a$ . Daher wird dann im **except**-Block ans Ende der Zeile der String `"None"` angehängt (die Zeile wird durch das Objekt `lst` repräsentiert).

Die Tabelle 4.9 auf Seite 120 enthält Beispiele für die Exponentiationen  $a^i \bmod 46$  sowie die Ordnungen  $\text{ord}_{46}(a)$

Der folgende Sage-Code erzeugt eine analoge Tabelle.

---

**Sage-Beispiel 4.5** Tabelle mit allen Potenzen  $a^i \bmod 46$  für  $a = 1, \dots, 23$  plus die Ordnung

---

von a

sage: m = 46

sage: euler\_phi(m)

22

sage: for a in xrange(1, 24):

....: lst = [power\_mod(a, i, m) for i in xrange(1, 24)]

....: try:

....: lst.append(multiplicative\_order(mod(a, m)))

....: except:

....: lst.append("None")

....: print lst

....:

[1, 1]

[2, 4, 8, 16, 32, 18, 36, 26, 6, 12, 24, 2, 4, 8, 16, 32, 18, 36, 26, 6, 12, 24, 2, 'None']

[3, 9, 27, 35, 13, 39, 25, 29, 41, 31, 1, 3, 9, 27, 35, 13, 39, 25, 29, 41, 31, 1, 3, 11]

[4, 16, 18, 26, 12, 2, 8, 32, 36, 6, 24, 4, 16, 18, 26, 12, 2, 8, 32, 36, 6, 24, 4, 'None']

[5, 25, 33, 27, 43, 31, 17, 39, 11, 9, 45, 41, 21, 13, 19, 3, 15, 29, 7, 35, 37, 1, 5, 22]

[6, 36, 32, 8, 2, 12, 26, 18, 16, 4, 24, 6, 36, 32, 8, 2, 12, 26, 18, 16, 4, 24, 6, 'None']

[7, 3, 21, 9, 17, 27, 5, 35, 15, 13, 45, 39, 43, 25, 37, 29, 19, 41, 11, 31, 33, 1, 7, 22]

[8, 18, 6, 2, 16, 36, 12, 4, 32, 26, 24, 8, 18, 6, 2, 16, 36, 12, 4, 32, 26, 24, 8, 'None']

[9, 35, 39, 29, 31, 3, 27, 13, 25, 41, 1, 9, 35, 39, 29, 31, 3, 27, 13, 25, 41, 1, 9, 11]

[10, 8, 34, 18, 42, 6, 14, 2, 20, 16, 22, 36, 38, 12, 28, 4, 40, 32, 44, 26, 30, 24, 10, 'None']

[11, 29, 43, 13, 5, 9, 7, 31, 19, 25, 45, 35, 17, 3, 33, 41, 37, 39, 15, 27, 21, 1, 11, 22]

[12, 6, 26, 36, 18, 32, 16, 8, 4, 2, 24, 12, 6, 26, 36, 18, 32, 16, 8, 4, 2, 24, 12, 'None']

[13, 31, 35, 41, 27, 29, 9, 25, 3, 39, 1, 13, 31, 35, 41, 27, 29, 9, 25, 3, 39, 1, 13, 11]

[14, 12, 30, 6, 38, 26, 42, 36, 44, 18, 22, 32, 34, 16, 40, 8, 20, 4, 10, 2, 28, 24, 14, 'None']

[15, 41, 17, 25, 7, 13, 11, 27, 37, 3, 45, 31, 5, 29, 21, 39, 33, 35, 19, 9, 43, 1, 15, 22]

[16, 26, 2, 32, 6, 4, 18, 12, 8, 36, 24, 16, 26, 2, 32, 6, 4, 18, 12, 8, 36, 24, 16, 'None']

[17, 13, 37, 31, 21, 35, 43, 41, 7, 27, 45, 29, 33, 9, 15, 25, 11, 3, 5, 39, 19, 1, 17, 22]

[18, 2, 36, 4, 26, 8, 6, 16, 12, 32, 24, 18, 2, 36, 4, 26, 8, 6, 16, 12, 32, 24, 18, 'None']

[19, 39, 5, 3, 11, 25, 15, 9, 33, 29, 45, 27, 7, 41, 43, 35, 21, 31, 37, 13, 17, 1, 19, 22]

[20, 32, 42, 12, 10, 16, 44, 6, 28, 8, 22, 26, 14, 4, 34, 36, 30, 2, 40, 18, 38, 24, 20, 'None']

[21, 27, 15, 39, 37, 41, 33, 3, 17, 35, 45, 25, 19, 31, 7, 9, 5, 13, 43, 29, 11, 1, 21, 22]

[22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 'None']

[23, 'None']

---

#### 4.18.4 Primitivwurzeln

Das Berechnen von Primitivwurzeln (siehe Kapitel 4.9, „Multiplikative Ordnung und Primitivwurzel“) geht in Sage sehr einfach: Sei  $n$  eine natürliche Zahl, dann kann mit dem Befehl `primitive_root(n)` eine Primitivwurzel der multiplikativen Gruppe  $(\mathbf{Z}/n\mathbf{Z})^*$  berechnet werden, sofern eine existiert. Ist  $n$  prim, dann ist das äquivalent zum Berechnen einer Primitivwurzel in  $\mathbf{Z}/n\mathbf{Z}$ .

Im folgenden berechnen wir die Primitivwurzeln einiger natürlicher Zahlen.

---

**Sage-Beispiel 4.6** Berechnen einer Primitivwurzel jeweils für eine Primzahl

---

```
sage: primitive_root(4)
3
sage: primitive_root(22)
13
sage: for p in primes(1, 50):
....:     print p, primitive_root(p)
....:
2 1
3 2
5 2
7 3
11 2
13 2
17 3
19 2
23 5
29 2
31 3
37 2
41 6
43 3
47 5
```

---

Ist  $p$  prim, dann hat  $\mathbf{Z}/p\mathbf{Z}$  mindestens eine Primitivwurzel.

Will man alle Primitivwurzeln von  $\mathbf{Z}/p\mathbf{Z}$  berechnen, und nicht nur irgend eine einzige von  $\mathbf{Z}/p\mathbf{Z}$ , dann kann man das mit der folgenden Funktion durchführen<sup>109</sup>.

---

**Sage-Beispiel 4.7** Funktion “enum\_PrimitiveRoots\_of\_an\_Integer” zur Berechnung aller Primitivwurzeln für eine gegebene Zahl

---

```
def enum_PrimitiveRoots_of_an_Integer(M):
    r"""
    Return all the primitive roots of the integer M (if possible).
    """
    try:
        g = primitive_root(M)
    except:
        return None
    targetOrder = euler_phi(M)
    L=[]
    # Stepping through all odd integers from 1 up to M, not including
    # M. So this loop only considers values of i where 1 <= i < M.
    for i in xrange(1,M,2):
        testGen = Mod(g^i,M)
        if testGen.multiplicative_order() == targetOrder:
            L.append(testGen.lift())
    # removing duplicates
    return Set(L)

# AA_Start -- Testcases for enum_PrimitiveRoots_of_an_Integer(M)
print "AA_Start -- Testcases for enum_PrimitiveRoots_of_an_Integer(M)"
M=10; print "1-----Testcase: M = %s" % M
LL = enum_PrimitiveRoots_of_an_Integer(M)
if LL==None:
    print M
else:
    print LL
M=8; print "2-----Testcase: M = %s" % M
# M=8 hat keine primitive root mod m. Checke, ob per try - except abgefangen.
LL = enum_PrimitiveRoots_of_an_Integer(M)
if LL==None:
    print M
else:
    print LL
M=17; print "3-----Testcase: M = %s" % M
LL = enum_PrimitiveRoots_of_an_Integer(M)
if LL==None:
    print M
else:
    print LL
# AA_End -- Testcases
```

```
OUTPUT:
AA_Start -- Testcases for enum_PrimitiveRoots_of_an_Integer(M)
1-----Testcase: M = 10
{3, 7}
2-----Testcase: M = 8
8
3-----Testcase: M = 17
{3, 5, 6, 7, 10, 11, 12, 14}
```

---

<sup>109</sup>Der folgende Code wurde als Sage-Skriptdatei erstellt und nicht-interaktiv ausgeführt. Deshalb gibt es in der Ausgabe keine Zeilen, die mit „sage:“ oder „.....“ anfangen wie in den Sage-Programmbeispielen bisher.

Das folgende Beispiel listet alle Primitivwurzeln der Primzahl 541 auf.

---

**Sage-Beispiel 4.8** Tabelle mit allen Primitivwurzeln der vorgegebenen Primzahl 541

---

```
sage: L=enum_PrimitiveRoots_of_an_Integer(541); L
{2, 517, 10, 523, 13, 14, 527, 528, 18, 531, 24, 539, 30, 37, 40, 51,
54, 55, 59, 62, 65, 67, 68, 72, 73, 77, 83, 86, 87, 91, 94, 96, 98,
99, 107, 113, 114, 116, 117, 126, 127, 128, 131, 132, 138, 150, 152,
153, 156, 158, 163, 176, 181, 183, 184, 195, 197, 199, 206, 208,
210, 213, 218, 220, 223, 224, 244, 248, 250, 257, 258, 259, 260,
261, 263, 267, 269, 270, 271, 272, 274, 278, 280, 281, 282, 283,
284, 291, 293, 297, 317, 318, 321, 323, 328, 331, 333, 335, 342,
344, 346, 357, 358, 360, 365, 378, 383, 385, 388, 389, 391, 403,
409, 410, 413, 414, 415, 424, 425, 427, 428, 434, 442, 443, 445,
447, 450, 454, 455, 458, 464, 468, 469, 473, 474, 476, 479, 482,
486, 487, 490, 501, 504, 511}
sage: len(L)
144
```

---

Mit etwas Programmieren kann man zählen, wie viele Primitivwurzeln es gibt für alle natürlichen Zahlen in einem gegebenen Zahlenbereich. Wir können das für alle Zahlen oder nur für die Primzahlen in diesem Bereich berechnen.

---

**Sage-Beispiel 4.9** Funktion “count\_PrimitiveRoots\_of\_an\_IntegerRange” zur Berechnung aller Primitivwurzeln für einen gegebenen Zahlenbereich

---

```
def count_PrimitiveRoots_of_an_IntegerRange(start, end, bPrimesOnly=True):
    r"""
    Compute all primitive roots of all numbers between start and end,
    inclusive, and count them.
    If the flag bPrimesOnly is True, it performs primality tests, so it
    allows us to count the number of primes from start to end, inclusive.
    If the flag bPrimesOnly is false, it additionally counts these even
    numbers which have no primitive root.
    """
    nCheckedNumb = 0
    nCheckedNumb_WithoutPrimitivRoots = 0
    nPrimitiveRoots = 0
    for n in xrange(start, end+1):
        if bPrimesOnly:
            if is_prime(n):
                nCheckedNumb += 1
                L = enum_PrimitiveRoots_of_an_Integer(n)
                nPrimitiveRoots += len(L)
        else:
            nCheckedNumb += 1
            L = enum_PrimitiveRoots_of_an_Integer(n)
            if L==None:
                nCheckedNumb_WithoutPrimitivRoots += 1
            else:
                nPrimitiveRoots += len(L)

    if bPrimesOnly:
        print "Found all %s " % nPrimitiveRoots + \
            " primitive roots of %s primes." % nCheckedNumb
    else:
        if nCheckedNumb_WithoutPrimitivRoots == 0:
            print "Found all %s " % nPrimitiveRoots + \
                "primitive roots of %s numbers." % nCheckedNumb
        else:
            print "Found all %s " % nPrimitiveRoots + \
                "primitive roots of %s numbers." % \
                (nCheckedNumb - nCheckedNumb_WithoutPrimitivRoots)
            print "(Total of numbers checked: %s, " % nCheckedNumb + \
                "Amount of numbers without primitive roots: %s)" % \
                nCheckedNumb_WithoutPrimitivRoots
```

---



Um zu sehen, wie lange unser Computer für diese Berechnung braucht, kann man den Sage-Befehl `time` verwenden.

---

**Sage-Beispiel 4.10** Funktion “`count_PrimitiveRoots_of_an_IntegerRange`”: Testfälle und Testausgaben

---

```
# BB_Start -- Testcases for count_PrimitiveRoots_of_an_IntegerRange(start, end, bPrimesOnly=True)
print "\n\nBB_Start -- Testcases for count_PrimitiveRoots_of_an_IntegerRange(start, end, True)"

print "\n1-----Testcase: (1, 500)"
time count_PrimitiveRoots_of_an_IntegerRange(1, 500)

print "\n2-----Testcase: (5, 6, False)"
time count_PrimitiveRoots_of_an_IntegerRange(5, 6, False)

print "\n3-----Testcase: (1, 500, False)"
time count_PrimitiveRoots_of_an_IntegerRange(1, 500, False)
# BB_End -- Testcases
```

OUTPUT:

```
BB_Start -- Testcases for count_PrimitiveRoots_of_an_IntegerRange(start, end, bPrimesOnly=True)
```

```
1-----Testcase: (1, 500)
```

```
Found all 8070 primitive roots of 95 primes.
```

```
Time: CPU 0.94 s, Wall: 0.97 s
```

```
2-----Testcase: (5, 6, False)
```

```
Found all 3 primitive roots of 2 numbers.
```

```
Time: CPU 0.00 s, Wall: 0.00 s
```

```
3-----Testcase: (1, 500, False)
```

```
Found all 11010 primitive roots of 170 numbers.
```

```
(Total of numbers checked: 500, Amount of numbers without primitive roots: 330)
```

```
Time: CPU 1.52 s, Wall: 1.59 s
```

---

Mit unserer selbst erstellten Funktion `enum_PrimitiveRoots_of_an_Integer` kann man alle Primitivwurzeln einer Primzahl  $p$  finden.

Die folgende Funktion zählt, wie viele Primitivwurzeln es innerhalb eines Primzahl-Bereiches gibt und gibt diese Primitivwurzeln jeweils aus.

Aus dieser Liste der Primitivwurzeln können wir jeweils die kleinste und größte Primitivwurzel pro  $\mathbf{Z}/p\mathbf{Z}$  bestimmen, als auch die Anzahl von Primitivwurzeln pro  $\mathbf{Z}/p\mathbf{Z}$  zählen.

---

**Sage-Beispiel 4.11** Funktion “`count_PrimitiveRoots_of_a_PrimesRange`” zur Berechnung aller Primitivwurzeln für ein gegebenes Intervall von Primzahlen

---

```
def count_PrimitiveRoots_of_a_PrimesRange(start, end):
    r"""
    Compute all primitive roots of all primes between start and end,
    inclusive. This uses a primes iterator.
    """
    nPrimes = 0
    nPrimitiveRoots = 0
    for p in primes(start, end+1):
        L = enum_PrimitiveRoots_of_an_Integer(p)
        print p, len(L)
        nPrimes += 1
        nPrimitiveRoots += len(L)
    print "Found all %s" % nPrimitiveRoots + " primitive roots of %s primes." % nPrimes

# CC_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)
print "\n\nBB_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)"
print "-----Testcase: (1, 1500)"
time count_PrimitiveRoots_of_a_PrimesRange(1, 1500)
# CC_End -- Testcases
```

OUTPUT:

```
CC_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)
-----Testcase: (1, 1500)
2 1
3 1
5 2
7 2
11 4
13 4
17 8
19 6
23 10
29 12
31 8
37 12
...
1483 432
1487 742
1489 480
1493 744
1499 636
Found all 62044 primitive roots of 239 primes.
Time: CPU 7.55 s, Wall: 7.85 s
```

---

Eine leicht geänderte Fassung unserer Funktion `count_PrimitiveRoots_of_a_PrimesRange` wird nun benutzt, um eine Liste (Datenbank) aller Primitivwurzeln für alle Primzahlen zwischen 1 und 100.000 zu erstellen.

---

**Sage-Beispiel 4.12** Code zur Erstellung einer Liste mit allen Primitivwurzeln für alle Primzahlen zwischen 1 und 100.000

---

```
start = 1
end = 10^5
fileName = "/scratch/mvngu/primroots.dat"
file = open(fileName, "w")
for p in primes(start, end+1):
    L = enum_PrimitiveRoots_of_an_Integer(p)
    print p, len(L)
    # Output to a file. The format is:
    # (1) the prime number p under consideration
    # (2) the number of primitive roots of Z/pZ
    # (3) all the primitive roots of Z/pZ
    file.write(str(p) + " " + str(len(L)) + " " + str(L) + "\n")
    file.flush()
file.close()
```

---

Es dauerte rund einen ganzen Tag auf der Maschine `sage.math`, um die Datei “primroots.dat” zu erstellen (erstellt im Juli 2009 von Minh Van Nguyen).

Auch dieser Code und die Funktion `enum_PrimitiveRoots_of_an_Integer` wurden in eine Sage-Skriptdatei eingefügt und nicht-interaktiv ausgeführt.

Die Datei “primroots.dat” enthält die Liste alle Primitivwurzeln für alle Primzahlen zwischen 1 und 100.000 inklusive. Dies ist eine sehr große Datei (ca. 1 GB unkomprimiert, ca. 285 MB komprimiert mit bzip2). Die Datei kann abgerufen werden unter <http://sage.math.washington.edu/home/mvngu/doc/primitive-roots/primroots.dat.bz2>.

Diese Datei “primroots.dat” wurde dann benutzt, um mit dem folgenden Code drei Grafiken zu erstellen.

---

**Sage-Beispiel 4.13** Code zur Erzeugung der Grafiken zur Verteilung der Primitivwurzeln

---

```
sage: # open a database file on primitive roots from 1 to 100,000
sage: file = open("/scratch/mvngu/primroots.dat", "r")
sage: plist = []      # list of all primes from 1 to 100,000
sage: nlist = []      # number of primitive roots modulo prime p
sage: minlist = []    # smallest primitive root modulo prime p
sage: maxlist = []    # largest primitive root modulo prime p
sage: for line in file:
....:     # get a line from the database file and tokenize it for processing
....:     line = line.strip().split(" ", 2)
....:     # extract the prime number p in question
....:     plist.append(Integer(line[0]))
....:     # extract the number of primitive roots modulo p
....:     nlist.append(Integer(line[1]))
....:     # extract the list of all primitive roots modulo p
....:     line = line[-1]
....:     line = line.replace("{", "")
....:     line = line.replace("}", "")
....:     line = line.split(", ")
....:     # sort the list in non-decreasing order
....:     line = [Integer(s) for s in line]
....:     line.sort()
....:     # get the smallest primitive root modulo p
....:     minlist.append(line[0])
....:     # get the largest primitive root modulo p
....:     maxlist.append(line[-1])
....:
sage: file.close() # close the database file
sage: # plot of number of primitive roots modulo p
sage: nplot = point2d(zip(plist, nlist), pointsize=1)
sage: nplot.axes_labels(["x", "y"])
sage: nplot
sage: # plot of smallest primitive root modulo prime p
sage: minplot = point2d(zip(plist, minlist), pointsize=1)
sage: minplot.axes_labels(["x", "y"])
sage: minplot
sage: # plot of largest primitive root modulo prime p
sage: maxplot = point2d(zip(plist, maxlist), pointsize=1)
sage: maxplot.axes_labels(["x", "y"])
sage: maxplot
```

---

Abbildung 4.2 gibt die Anzahl der Primitivwurzeln für jede Primzahl zwischen 1 und 100.000 aus. Die  $x$ -Achse repräsentiert die Primzahlen 1 bis 100.000, die  $y$ -Achse gibt die Anzahl der Primitivwurzeln pro Primzahl aus.

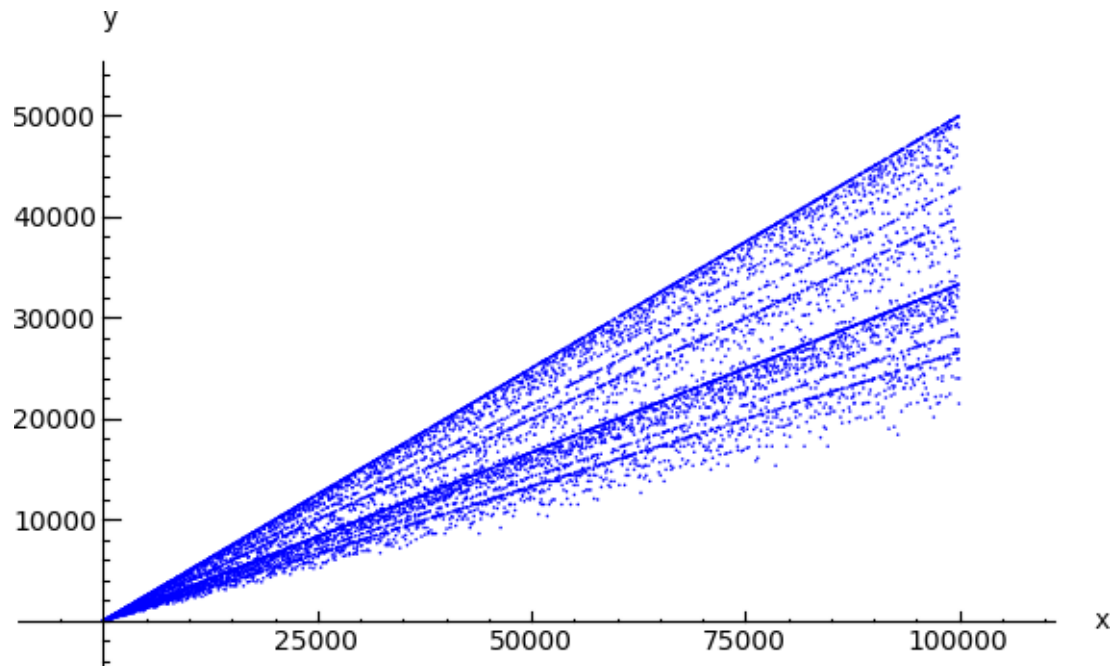


Abbildung 4.2: Die Anzahl der Primitivwurzeln für alle Primzahlen zwischen 1 und 100.000

Abbildung 4.3 gibt die kleinste Primitivwurzel von jeder Primzahl zwischen 1 und 100.000 aus. Die  $x$ -Achse repräsentiert die Primzahlen 1 bis 100.000, die  $y$ -Achse gibt die kleinste Primitivwurzel pro Primzahl aus.

Abbildung 4.4 zeigt die entsprechende Graphik mit der größten Primitivwurzel zu jeder Primzahl im obigen Intervall.

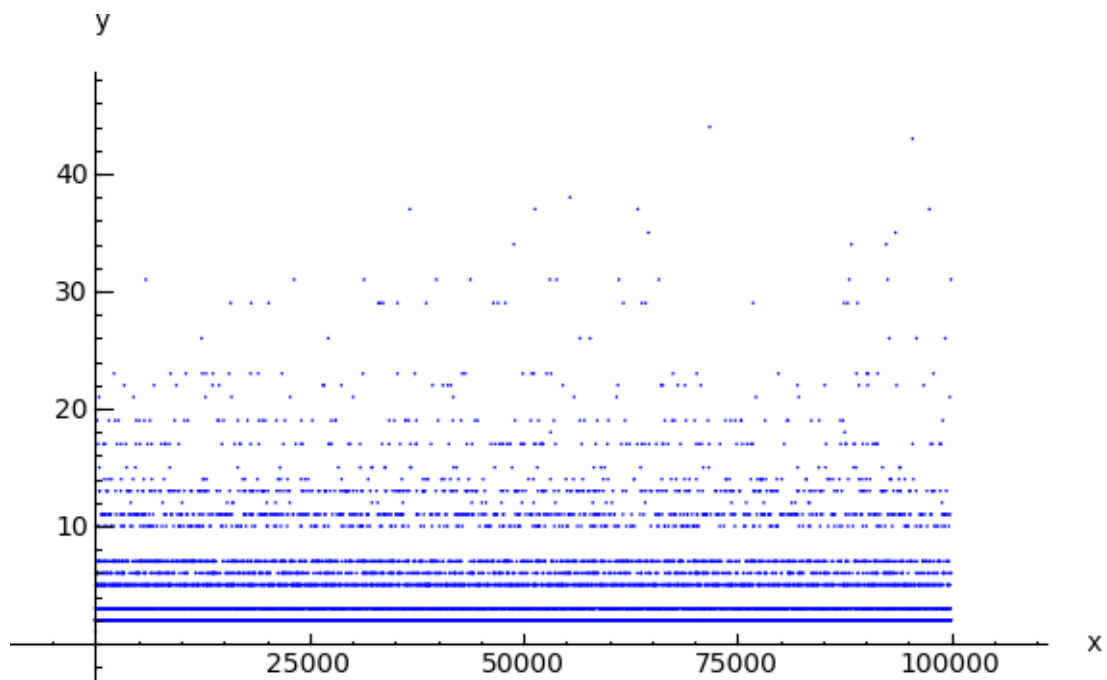


Abbildung 4.3: Die kleinste Primitivwurzel von jeder Primzahl zwischen 1 und 100.000

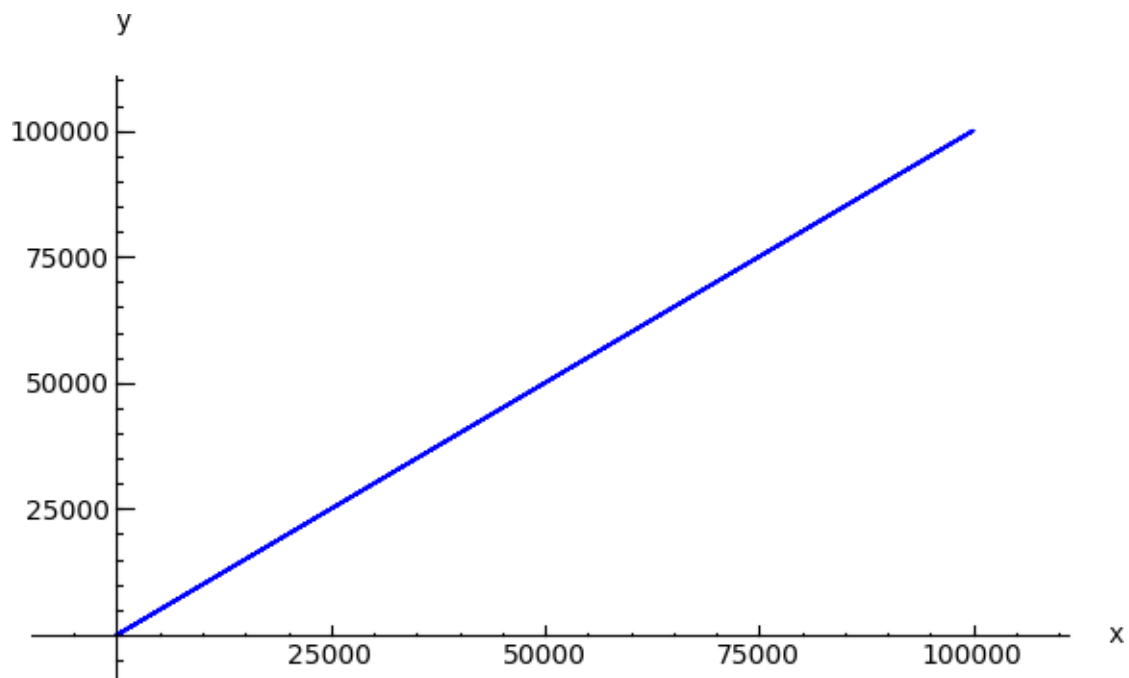


Abbildung 4.4: Die größte Primitivwurzel von jeder Primzahl zwischen 1 und 100.000

### 4.18.5 RSA-Beispiele mit Sage

In diesem Abschnitt sind die Sage-Quelltexte für die einfachen RSA-Beispiele im Kapitel 4.13 (“Das RSA-Verfahren mit konkreten Zahlen”) angegeben.

#### Beispiel auf Seite 143:

Die RSA-Exponentiation  $M^{37} \pmod{3713}$  für die Nachricht  $M = 120$  kann mit Sage folgendermaßen ausgeführt werden:

```
sage: power_mod(120, 37, 3713)
1404
```

#### Beispiel auf Seite 144:

Die Faktorisierung von  $J(256.027) = 255.016 = 2^3 * 127 * 251$  kann mit Sage folgendermaßen durchgeführt werden:

---

#### Sage-Beispiel 4.14 Faktorisierung einer Zahl

---

```
sage: factor(255016)
2^3 * 127 * 251
```

---

#### Beispiel auf Seite 144:

RSA-Verschlüsselung mit Sage:

---

#### Sage-Beispiel 4.15 RSA-Verschlüsselung durch modulare Exponentiation einer Zahl (als Nachricht)

---

```
sage: A = [82, 83, 65, 32, 119, 111, 114, 107, 115, 33]
sage: e = 65537; m = 256027
sage: [power_mod(a, e, m) for a in A]
[212984, 25546, 104529, 31692, 248407, 100412, 54196, 100184, 58179, 227433]
```

---

#### Beispiel auf Seite 145:

RSA-Verschlüsselung mit Sage:

```
sage: A = [21075, 16672, 30575, 29291, 29473]
sage: e = 65537; m = 256027
sage: [power_mod(a, e, m) for a in A]
[158721, 137346, 37358, 240130, 112898]
```

#### Beispiel auf Seite 145:

RSA-Verschlüsselung mit Sage:

```
sage: A = [82083, 65032, 119111, 114107, 115033]
sage: e = 65537; m = 256027
sage: [power_mod(a, e, m) for a in A]
[198967, 51405, 254571, 115318, 14251]
```

#### 4.18.6 Wie viele RSA-Schlüssel gibt es innerhalb eines Modulo-Bereiches?

Die RSA-Verschlüsselung wurde beschrieben in Abschnitt 4.10.2 (“Funktionsweise des RSA-Verfahrens”). Schritt 1 bis 3 definieren die Schlüsselerzeugung, Schritt 4 und 5 stellen die eigentliche Verschlüsselung dar:

1. Wähle zwei unterschiedliche Primzahlen  $p$  und  $q$  und berechne  $n = p * q$ .  
Der Wert  $n$  wird RSA-Modul genannt.
2. Wähle ein zufälliges  $e \in \{2, \dots, n - 1\}$ , so dass gilt:  
 $e$  ist relativ prim zu  $J(n) = (p - 1) * (q - 1)$ .  
Danach kann man  $p$  und  $q$  „wegwerfen“.
3. Wähle  $d \in \{1, \dots, n - 1\}$  mit  $e * d \equiv 1 \pmod{J(n)}$ ,  
d.h.  $d$  ist die multiplikative Inverse von  $e$  modulo  $J(n)$ . Dann kann man  $J(n)$  „wegwerfen“.  
 $\rightarrow (n, e)$  ist der öffentliche Schlüssel  $P$ .  
 $\rightarrow (n, d)$  ist der private Schlüssel  $S$  (nur  $d$  muss man geheim halten).
4. Zur Verschlüsselung wird die Nachricht als (binäre) Ziffernfolge geschrieben. Diese Ziffernfolge wird dann so in gleich lange Teilfolgen aufgeteilt, dass jede Teilfolge eine Zahl kleiner als  $n$  darstellt.
5. Vorgang der Verschlüsselung auf dem Klartext (bzw. auf Teilen davon)  $M \in \{1, \dots, n - 1\}$ :

$$C = E((n, e); M) := M^e \pmod{n}.$$

Standardmäßig versucht man, einen mit RSA verschlüsselten Geheimtext  $C$  dadurch zu knacken, dass man den öffentlichen Schlüssel des Empfängers betrachtet und versucht,  $n$  zu faktorisieren. Hat man das erreicht, geht man wieder durch die Schritte 2 und 3 und erzeugt den privaten Schlüssel  $e$ , den man zum Entschlüsseln des Geheimtextes braucht.

Gemäß dem „Primzahlsatz“ (beschrieben in Abschnitt 3.7.2 „Die Anzahl der Primzahlen  $PI(x)$ “) bewegt sich die Anzahl der Primzahlen  $PI(x)$  asymptotisch gegen  $x/\ln(x)$ . Für ein gegebenes  $n$  gibt es also ca.  $n/\ln(n)$  mögliche Werte für die Primzahl  $p$ .

Will man nicht faktorisieren, sondern stellt sich eine Frage ähnlich wie bei den klassischen Verschlüsselungsverfahren, kann man herausfinden wollen: Wie viele verschiedene private Schlüssel  $(n, d)$  gibt es für einen bestimmten Bereich der Schlüsselgröße  $n \in [a, b]$ ?

Das Sage-Beispielprogramm 4.16 unten definiert die Funktion `count_Number_of_RSA_Keys`, die diese Frage konkret beantworten kann (wenn der Modulus nicht zu groß ist).<sup>110</sup>

Weil es mehr private Schlüssel  $(n, d)$  innerhalb eines größeren Bereiches von Werten für  $n$  gibt, ist das Brute-Force-Faktorisieren viel effizienter als das Durchprobieren aller möglichen Schlüssel.

<sup>110</sup> a) Der Aufruf `sage: count_Number_of_RSA_Keys(100, 1000)` bedeutet, dass man das Intervall  $[100, 1000]$  für  $n$  betrachtet.  $n$  war definiert durch die beiden Primzahlen  $p, q : n = p * q$ . Daher kann eine Primzahl höchstens den Wert 500 annehmen, weil  $2 * 500 = 1000$ .

Die Anzahl möglicher Primzahl-Kombinationen beträgt: `comb = 258`.

Die Anzahl der Primzahlen im gegebenen Bereich beträgt: 143.

Die Anzahl der privaten Schlüssel beträgt: 34.816.

b) Der Aufruf `sage: count_Number_of_RSA_Keys(100, 100, True)` hat die folgende Ausgabe:

- Number of private keys for modulus in a given range: 0

- Number of primes in a given range: 0

Der Grund dafür ist, dass wir damit nur  $n = 100$  betrachten, und 100 ist nicht semiprim, d.h. 100 ist nicht das Produkt von genau zwei Primzahlen.



---

**Sage-Beispiel 4.16** Wie viele RSA-Schlüssel gibt es, wenn man den Bereich für die Schlüsselgröße  $n$  kennt?

---

```
def count_Number_of_RSA_Keys(start, end, Verbose=False):
    r"""
    How many private RSA keys (n,d) exist, if only modulus N is given, and start <= N <= end?
    (prime_range(u,o) delivers all primes >=u und < o).
    """
    a = start
    b = end
    s = 0
    comb = 0
    for p in prime_range(1, b/2+1):
        for q in prime_range(p + 1, b/2+1):
            if a <= p * q and p * q <= b:
                comb = comb + 1
                s = s + (euler_phi(euler_phi(p * q))-1)
                if Verbose:
                    print "p=%s, " % p + "q=%s, " % q + "s=%s" % s
    print "Number of private keys for modulus in a given range: %s" % s + " (comb=%s), " % comb

    # Just for comparison: How many primes are in this range?
    s = 0
    for p in prime_range(a, b+1):
        if Verbose:
            print "a=%s, " % a + "b=%s, " % b + "p=%s" % p
        s = s + 1
    print "Number of primes in a given range: %s" % s

print "\n\nDD_Start -- Testcases for count_Number_of_RSA_Keys(start, end)"
print "\n-----Testcase: (100, 1000) [Should deliver 34.816]"
time count_Number_of_RSA_Keys(100, 1000)
print "\n-----Testcase: (100, 107, True) [Should deliver 23]"
time count_Number_of_RSA_Keys(100, 107, True)
u = 10^3; o = 10^4;
print "\n-----Testcase: (%s, " % u + "%s) [Should deliver 3.260.044]" % o
time count_Number_of_RSA_Keys(u, o)
```

OUTPUT:

```
DD_Start -- Testcases for count_Number_of_RSA_Keys(start, end)

-----Testcase: (100, 1000) [Should deliver 34.816]
Number of private keys for modulus in a given range: 34816 (comb=258),
Number of primes in a given range: 143
Time: CPU 0.03 s, Wall: 0.04 s

-----Testcase: (100, 107, True) [Should deliver 23]
p=2, q=53, s=23
Number of private keys for modulus in a given range: 23 (comb=1),
a=100, b=107, p=101
a=100, b=107, p=103
a=100, b=107, p=107
Number of primes in a given range: 3
Time: CPU 0.00 s, Wall: 0.00 s

-----Testcase: (1000, 10000) [Should deliver 3.260.044]
Number of private keys for modulus in a given range: 3260044 (comb=2312),
Number of primes in a given range: 1061
Time: CPU 0.63 s, Wall: 0.66 s
```

---

## 4.19 Anhang: Liste der in diesem Kapitel formulierten Definitionen und Sätze

	Kurzbeschreibung	Seite
Definition 4.3.1	Primzahlen	99
Definition 4.3.2	Zusammengesetzte Zahlen	99
Satz 4.3.1	Teiler von zusammengesetzten Zahlen	100
Satz 4.3.2	Erster Hauptsatz der elementaren Zahlentheorie	100
Definition 4.4.1	Teilbarkeit	101
Definition 4.4.2	Restklasse $r$ modulo $m$	101
Definition 4.4.3	restgleich oder kongruent	102
Satz 4.4.1	Kongruenz mittels Differenz	102
Satz 4.6.1	Multiplikative Inverse (Existenz)	106
Satz 4.6.2	Erschöpfende Permutation	108
Satz 4.6.3	Gestaffelte Exponentiation mod $m$	110
Definition 4.7.1	$\mathbb{Z}_n$	112
Definition 4.7.2	$\mathbb{Z}_n^*$	113
Satz 4.7.1	Multiplikative Inverse in $\mathbb{Z}_n^*$	113
Definition 4.8.1	Euler-Funktion $J(n)$	115
Satz 4.8.1	$J(p)$	115
Satz 4.8.2	$J(p * q)$	115
Satz 4.8.3	$J(p_1 * \cdots * p_k)$	115
Satz 4.8.4	$J(p_1^{e_1} * \cdots * p_k^{e_k})$	115
Satz 4.8.5	Kleiner Satz von Fermat	116
Satz 4.8.6	Satz von Euler-Fermat	116
Definition 4.9.1	Multiplikative Ordnung $\text{ord}_m(a)$	118
Definition 4.9.2	Primitivwurzel von $m$	118
Satz 4.9.1	Ausschöpfung des Wertebereiches	120

# Literaturverzeichnis

- [Agrawal2002] M. Agrawal, N. Kayal, N. Saxena,  
*PRIMES in P*, August 2002, Korrigierte Fassung:  
[http://www.cse.iitk.ac.in/~manindra/algebra/primality\\_v6.pdf](http://www.cse.iitk.ac.in/~manindra/algebra/primality_v6.pdf)  
Siehe auch die Seite „The AKS „PRIMES in P” Algorithm Resource”: <http://fatphil.org/maths/AKS/>.
- [Bartholome1996] A. Bartholomé, J. Rung, H. Kern,  
*Zahlentheorie für Einsteiger*, Vieweg 1995, 2. Auflage 1996.
- [Bauer1995] Friedrich L. Bauer,  
*Entzifferte Geheimnisse*, Springer, 1995.
- [Bauer2000] Friedrich L. Bauer,  
*Decrypted Secrets*, Springer 1997, 2nd edition 2000.
- [Bernstein2001] D. J. Bernstein,  
*Circuits for integer factorization: a proposal*,  
<http://cr.yp.to/papers/nfscircuit.ps>  
<http://cr.yp.to/djb.html>.
- [Beutelspacher1996] Albrecht Beutelspacher,  
*Kryptologie*, Vieweg 1987, 5. Auflage 1996.
- [Bourseau2002] F. Bourseau, D. Fox, C. Thiel,  
*Vorzüge und Grenzen des RSA-Verfahrens*,  
In: Datenschutz und Datensicherheit (DuD) 26/2002, S. 84-89 (s. [www.dud.de](http://www.dud.de)), <http://www.secorvo.de/publikationen/rsa-grenzen-fox-2002.pdf>.
- [Brands2002] Gilbert Brands,  
*Verschlüsselungsalgorithmen – Angewandte Zahlentheorie rund um Sicherheitsprotokolle*, Vieweg, 2002.
- [BSI2002] BSI (Bundesamt für Sicherheit in der Informationstechnik),  
*Empfehlungen zur Wahl der Schlüssellängen*, Bonn, 9.9.2002,  
(Geeignete Kryptoalgorithmen zur Erfüllung der Anforderungen nach §17 (1) SigG vom 22. Mai 2001 in Verbindung mit Anlage 1, I 2, SigV vom 22. November 2001),  
<http://www.bsi.bund.de/esig/basics/techbas/krypto/bund02v7.pdf>  
Eine Stellungnahme zu diesen Empfehlungen:  
<http://www.secorvo.de/publikat/stellungnahme-algorithmenempfehlung-020307.pdf> .
- [Buchmann2004] Johannes Buchmann,  
*Einführung in die Kryptographie*, Springer, 3. Auflage, 2004.

- [Buhler1993] J.P. Buhler, H.W. Lenstra, C. Pomerance,  
*Factoring integers with the number field sieve*,  
 In: A.K. Lenstra, H.W. Lenstra (Hrsg.): The Development of the Number Field Sieve,  
 Lecture Notes in Mathematics, Vol. 1554, Springer, Heidelberg 1993, S. 50–94.
- [Eckert2003] Claudia Eckert,  
*IT-Sicherheit: Konzepte-Verfahren-Protokolle*, Oldenbourg 2001, 2. Auflage 2003.
- [Ertel2001] Wolfgang Ertel,  
*Angewandte Kryptographie*, Fachbuchverlag Leipzig FV 2001.
- [Graham1994] Graham, Knuth, Patashnik,  
*Concrete Mathematics, a Foundation of Computer Science*,  
 Addison Wesley 1989, 6th printing 1994.
- [Kippenhahn1997] Rudolph Kippenhahn,  
*Verschlüsselte Botschaften – Geheimschrift, Enigma und Chipkarte*, Rowohlt, 1997.
- [Kippenhahn1999] Rudolph Kippenhahn,  
*Code Breaking – A History and Exploration*, Constable, 1999.
- [Kleijung2010] Thorsten Kleijung et al.  
*Factorization of a 768-bit RSA modulus*,  
<http://eprint.iacr.org/2010/006.pdf>.
- [Knuth1998] Donald E. Knuth,  
*The Art of Computer Programming, vol 2: Seminumerical Algorithms*,  
 Addison-Wesley, 2nd edition 1998.
- [Lenstra1993] A. Lenstra, H. Lenstra:  
*The development of the Number Field Sieve*,  
 Lecture Notes in Mathematics 1554, Springer, New York 1993
- [Lenstra1999] Arjen K. Lenstra, Eric R. Verheul,  
*Selecting Cryptographic Key Sizes (1999)*,  
 Journal of Cryptology: the journal of the International Association for Cryptologic  
 Research,  
<http://www.cryptosavvy.com/cryptosizes.pdf>.
- [Lenstra2002] Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, Eran Tromer,  
*Analysis of Bernstein's Factorization Circuit*,  
<http://www.cryptosavvy.com/mesh.pdf>.
- [Menezes2001] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone  
*Handbook of Applied Cryptography*, CRC Press 1997, 5th printing 2001.
- [Pfleeger1997] Charles P. Pfleeger,  
*Security in Computing*, Prentice-Hall, 2nd edition 1997.
- [Pomerance1984] C. Pomerance,  
*The quadratic sieve factoring algorithm*,  
 In: G.R. Blakley, D. Chaum (Hrsg.): Proceedings of Crypto '84, LNCS 196, Springer  
 Berlin 1995, S. 169-182.

- [RSA Security 2002] RSA Security,  
*Has the RSA algorithm been compromised as a result of Bernstein's Paper?*,  
 8. April 2002,  
<http://www.rsasecurity.com/>.
- [SchneiderM2004] Matthias Schneider,  
*Analyse der Sicherheit des RSA-Algorithmus.*  
*Mögliche Angriffe, deren Einfluss auf sichere Implementierungen und ökonomische Konsequenzen,*  
 Diplomarbeit an der Universität Siegen, 2004.
- [Schneier1996] Bruce Schneier,  
*Applied Cryptography, Protocols, Algorithms, and Source Code in C,*  
 Wiley and Sons 1994, 2nd edition 1996.
- [Schwenk2002] Jörg Schwenk,  
*Sicherheit und Kryptographie im Internet,* Vieweg 2002.
- [Sedgewick1990] Robert Sedgewick,  
*Algorithms in C,* Addison-Wesley, 1990.
- [Shamir2003] Adi Shamir, Eran Tromer,  
*Factoring Large Numbers with the TWIRL Device,* Januar 2003,  
<http://www.wisdom.weizmann.ac.il/~tromer/>.
- [Shamir2003a] Adi Shamir, Eran Tromer,  
*On the Cost of Factoring RSA-1024,* RSA Laboratories CryptoBytes Volume 6, No. 2,  
 Summer 2003, S. 11-20,  
[http://www.rsasecurity.com/rsalabs/cryptobytes/CryptoBytes\\_August\\_2003.pdf](http://www.rsasecurity.com/rsalabs/cryptobytes/CryptoBytes_August_2003.pdf).
- [Silverman2000] Robert D. Silverman:  
*A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths*  
 In: RSA Laboratories Bulletin, No. 13, April 2000, S. 1-22
- [Stinson1995] Douglas R. Stinson,  
*Cryptography - Theory and Practice,* CRC Press, 1995.
- [Weis2003] Rüdiger Weis, Stefan Lucks, Andreas Bogk,  
*Sicherheit von 1024 bit RSA-Schlüsseln gefährdet,*  
 In: Datenschutz und Datensicherheit (DuD) 6/2003, S. 360-362 (s. [www.dud.de](http://www.dud.de))  
 Der Artikel erläutert Details zum TWIRL-Device.
- [Welschenbach2001] Welschenbach, Michael,  
*Kryptographie in C und C++,* Springer 2001.
- [Wiles1994] Wiles, Andrew,  
*Modular elliptic curves and Fermat's Last Theorem,*  
 In: Annals of Mathematics 141 (1995).
- [Wolfenstetter1998] Albrecht Beutelspacher, Jörg Schwenk, Klaus-Dieter Wolfenstetter,  
*Moderne Verfahren in der Kryptographie,* Vieweg 1995, 2. Auflage 1998.
- [Yan2000] Song Y. Yan,  
*Number Theory for Computing,* Springer, 2000.

## Web-Links

1. Fibonacci-Seite von Ron Knott,  
Hier dreht sich alles um Fibonacci-Zahlen.  
<http://www.mcs.surrey.ac.uk/personal/R.Knott/Fibonacci/fib.html>
2. CrypTool,  
E-Learning-Freeware zur Veranschaulichung von Kryptographie und Kryptoanalyse,  
<http://www.cryptool.de>,  
<http://www.cryptool.org>,  
<http://www.cryptool.com>
3. Mathematica,  
Kommerzielles Mathematik-Paket  
<http://www.wolfram.com>
4. LiDIA,  
Umfangreiche Bibliothek mit zahlentheoretischen Funktionen und dem Interpreter LC.  
Wird nicht weiter entwickelt.  
<http://www.informatik.tu-darmstadt.de/TI/LiDIA>
5. BC,  
Interpreter mit zahlentheoretischen Funktionen  
<http://www.gnu.org/software/bc/bc.html>
6. Pari-GP,  
Hervorragender, schneller und freier Interpreter mit zahlentheoretischen Funktionen.  
<http://pari.math.u-bordeaux.fr/>  
<http://en.wikipedia.org/wiki/PARI/GP>  
Ressourcen zu PARI/GP auf der Webseite von Karim Belabas:  
<http://www.math.u-bordeaux.fr/~belabas/pari/>
7. Erst nach Vollendung dieses Artikels wurde mir die Web-Seite von Herrn Münchenbach bekannt, die interaktiv und didaktisch sehr ausgereift die grundlegenden Denkweisen der Mathematik anhand der elementaren Zahlentheorie nahebringt. Sie entstand für ein Unterrichtsprojekt der 11. Klasse des Technischen Gymnasiums (leider nur in Deutsch verfügbar):  
<http://www.hydrargyrum.de/kryptographie>
8. Seite des Beauftragten für die Lehrplanentwicklung des Fachs Informatik an der gymnasialen Oberstufe des Landes Saarland. Hier befindet sich eine Sammlung von Texten und Programmen (in Java), die aus didaktischen Überlegungen entstand (alles leider nur in Deutsch verfügbar).  
<http://www.saar.de/~awa/kryptolo.htm>
9. BSI,  
Bundesamt für Sicherheit in der Informationstechnik  
<http://www.bsi.bund.de>
10. Faktorisierungsrekorde und Challenges,  
<http://www.crypto-world.com/>  
<http://www.crypto-world.com/FactorWorld.html>, Webseite von Scott Contini

<http://www.loria.fr/~zimmerma/records/factor.html>

[http://www.tutorgig.com/ed/RSA\\_number](http://www.tutorgig.com/ed/RSA_number)

<http://www.uni-bonn.de/Aktuelles/Pressemitteilungen/pm02/pm035-02.html>

[http://www.ercim.org/publication/Ercim\\_News/enw49/franke.html](http://www.ercim.org/publication/Ercim_News/enw49/franke.html)

<http://www.loria.fr/~zimmerma/records/rsa160>

<http://www.rsa.com/rsalabs/node.asp?id=2092>

11. Das Cunningham-Projekt,

<http://www.cerias.purdue.edu/homes/ssw/cun/>

12. Sage,

Ausgezeichnetes Open-Source Computer-Algebra-System, basierend auf Python als Skript-Sprache. Damit sind die Code-Beispiele in diesem Kapitel erstellt. Vergleiche die Einführung in Kapitel A.5.

<http://www.sagemath.org/>

[http://en.wikipedia.org/wiki/Sage\\_%28mathematics\\_software%29](http://en.wikipedia.org/wiki/Sage_%28mathematics_software%29)

## Dank

Ich möchte hier die Gelegenheit nutzen, den folgenden Personen ganz herzlich zu danken:

- Hr. Henrik Koy für das anregende und sehr konstruktive Korrekturlesen und für die vielen Verbesserungen der ersten Version dieses Artikels, und für die Hilfen bei TeX, ohne die dieser Artikel nie in dieser Form erschienen wäre.  
Außerdem hat Hr. Koy in seiner Freizeit im CrypTool-Programm die komplexe Dialogbox zum RSA-Kryptosystem designed und die dahinterliegende Logik entwickelt, mit der die RSA-Beispiele dieses Artikels nachvollzogen werden können.
- Jörg Cornelius Schneider für die engagierte Unterstützung bei TeX und die mannigfaltigen Hilfen bei allen Arten von Programmier- und Design-Problemen.
- Dr. Georg Illies für den Hinweis auf Pari-GP.
- Lars Fischer für seine Hilfe bei schnellem Pari-GP-Code für Primitivwurzeln.
- Minh Van Nguyen aus Australien für seine immer schnelle, kompetente und ausführliche Hilfe bei den Sage-Code-Beispielen in diesem Kapitel.

## Kapitel 5

# Die mathematischen Ideen hinter der modernen Kryptographie<sup>1</sup>

(Oyono R./ Esslinger B./ Schneider J., Sep. 2000; Updates Nov. 2000, Feb. 2003, Apr. 2007, März 2010)

*Georg Christoph Lichtenberg<sup>2</sup>:*

Ich weiß nicht, ob es besser wird, wenn wir es ändern,  
aber ich weiß, dass wir es ändern müssen, wenn es besser werden soll.

*Anmerkung von Unbekannt (Radio) dazu:*

Und Gott gebe den Akteuren bei der notwendigen Umsetzung  
das Wissen, die Weisheit und das Verantwortungsbewusstsein,  
zwischen Aktionismus, Selbstdarstellung und planvollem Handeln  
zu unterscheiden – und ihr Wissen auch anzuwenden.

### 5.1 Einwegfunktionen mit Falltür und Komplexitätsklassen

Eine **Einwegfunktion** ist eine effizient zu berechnende Funktion, deren Umkehrung jedoch nur mit extrem hohem Rechenaufwand – jedoch praktisch unmöglich – zu berechnen ist.

Etwas genauer formuliert: Eine Einwegfunktion ist eine Abbildung  $f$  einer Menge  $X$  in eine Menge  $Y$ , so dass  $f(x)$  für jedes Element  $x$  von  $X$  leicht zu berechnen ist, während es für (fast) jedes  $y$  aus  $Y$  praktisch unmöglich ist, ein Urbild  $x$  (d.h. ein  $x$  mit  $f(x) = y$ ) zu finden.

Ein alltägliches Beispiel für eine Einwegfunktion ist ein Telefonbuch: die auszuführende Funktion ist die, einem Namen die entsprechende Telefonnummer zuzuordnen. Da die Namen alphabetisch geordnet sind, ist diese Zuordnung einfach auszuführen. Aber ihre Invertierung, also die Zuordnung eines Namens zu einer gegebenen Nummer, ist offensichtlich schwierig, wenn

---

<sup>1</sup>Mit dem Lernprogramm **ZT** können Sie spielerisch einige der hier besprochenen Verfahren (RSA, Rabin, DH, ElGamal) nachvollziehen (siehe Lern-Kapitel 4.2 und 4.3, Seiten 9-17/17).

ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

<sup>2</sup>Georg Christoph Lichtenberg, deutscher Schriftsteller und Physiker (1742-1799),  
(siehe auch: [http://de.wikipedia.org/wiki/Georg\\_Christoph\\_Lichtenberg](http://de.wikipedia.org/wiki/Georg_Christoph_Lichtenberg))



man nur ein Telefonbuch zur Verfügung hat.

Einwegfunktionen spielen in der Kryptographie eine entscheidende Rolle. Fast alle kryptographischen Begriffe kann man durch Verwendung des Begriffs Einwegfunktion umformulieren. Als Beispiel betrachten wir die Public-Key-Verschlüsselung (asymmetrische Kryptographie):

Jedem Teilnehmer  $T$  des Systems wird ein privater Schlüssel  $d_T$  und ein sogenannter öffentlicher Schlüssel  $e_T$  zugeordnet. Dabei muss die folgende Eigenschaft (Public-Key-Eigenschaft) gelten:

Für einen Gegner, der den öffentlichen Schlüssel  $e_T$  kennt, ist es praktisch unmöglich, den privaten Schlüssel  $d_T$  zu bestimmen.

Zur Konstruktion nützlicher Public-Key-Verfahren sucht man also eine Einwegfunktion, die in einer Richtung „einfach“ zu berechnen, die in der anderen Richtung jedoch „schwer“ (praktisch unmöglich) zu berechnen ist, solange eine bestimmte zusätzliche Information (Falltür) nicht zur Verfügung steht. Mit der zusätzlichen Information kann die Umkehrung effizient gelöst werden. Solche Funktionen nennt man **Einwegfunktionen mit Falltür** (trapdoor one-way function). Im obigen Fall ist  $d_T$  die Falltür-Information.

Dabei bezeichnet man ein Problem als „einfach“, wenn es in polynomialer Zeit als Funktion der Länge der Eingabe lösbar ist, d.h. wenn es so gelöst werden kann, dass der Zeitaufwand sich als polynomiale Funktion in Abhängigkeit der Länge der Eingabe darstellen lässt. Wenn die Länge der Eingabe  $n$  Bits beträgt, so ist die Zeit der Berechnung der Funktion proportional zu  $n^a$ , wobei  $a$  eine Konstante ist. Man sagt, dass die Komplexität solcher Probleme  $O(n^a)$  beträgt (Landau- oder Big-O-Notation).

Vergleicht man 2 Funktionen  $2^n$  und  $n^a$ , wobei  $a$  eine Konstante ist, dann gibt es immer einen Wert für  $n$ , ab dem für alle weiteren  $n$  gilt:  $n^a < 2^n$ . Die Funktion  $n^a$  hat eine geringere Komplexität. Z.B. für  $a = 5$  gilt: ab der Länge  $n = 23$  ist  $2^n > n^5$  und danach wächst  $2^n$  auch deutlich schneller [( $2^{22} = 4.194.304$ ,  $22^5 = 5.153.632$ ), ( $2^{23} = 8.388.608$ ,  $23^5 = 6.436.343$ ), ( $2^{24} = 16.777.216$ ,  $24^5 = 7.962.624$ )].

Der Begriff „praktisch unmöglich“ ist etwas schwammiger. Allgemein kann man sagen, ein Problem ist nicht effizient lösbar, wenn der zu seiner Lösung benötigte Aufwand schneller wächst als die polynomiale Zeit als Funktion der Größe der Eingabe. Wenn beispielsweise die Länge der Eingabe  $n$  Bits beträgt und die Zeit zur Berechnung der Funktion proportional zu  $2^n$  ist, so gilt gegenwärtig: die Funktion ist für  $n > 80$  praktisch nicht zu berechnen.

Die Entwicklung eines praktisch einsetzbaren Public-Key-Verfahrens hängt daher von der Entdeckung einer geeigneten Einwegfunktion mit Falltür ab.

Um Ordnung in die verwirrende Vielfalt von möglichen Problemen und ihre Komplexitäten zu bringen, fasst man Probleme mit ähnlicher Komplexität zu Klassen zusammen.

Die wichtigsten Komplexitätsklassen sind die Klassen **P** und **NP**:

- Die Klasse **P**: Zu dieser Klasse gehören diejenigen Probleme, die mit polynomialem Zeitaufwand lösbar sind.
- Die Klasse **NP**: Bei der Definition dieser Klasse betrachten wir nicht den Aufwand zur Lösung eines Problems, sondern den Aufwand zur Verifizierung einer gegebenen Lösung. Die Klasse **NP** besteht aus denjenigen Problemen, bei denen die Verifizierung einer gegebenen Lösung mit polynomialem Zeitaufwand möglich ist. Dabei bedeutet der Begriff **NP** „nichtdeterministisch“ polynomial und bezieht sich auf ein Berechnungsmodell, d.h. auf einen nur in der Theorie existierenden Computer, der richtige Lösungen nichtdeterministisch „raten“ und dies dann in polynomialer Zeit verifizieren kann.

Die Klasse  $\mathbf{P}$  ist in der Klasse  $\mathbf{NP}$  enthalten. Ein berühmtes offenes Problem ist die Frage, ob  $\mathbf{P} \neq \mathbf{NP}$  gilt oder nicht, d.h. ob  $\mathbf{P}$  eine echte Teilmenge ist oder nicht. Eine wichtige Eigenschaft der Klasse  $\mathbf{NP}$  ist, dass sie auch sogenannte „ $\mathbf{NP}$ -vollständige“ Probleme enthält. Dies sind Probleme, welche die Klasse  $\mathbf{NP}$  im Folgenden Sinne vollständig repräsentieren: Wenn es einen „guten“ Algorithmus für ein solches Problem gibt, dann existieren für alle Probleme aus  $\mathbf{NP}$  „gute“ Algorithmen. Insbesondere gilt: wenn auch nur ein vollständiges Problem in  $\mathbf{P}$  läge, d.h. wenn es einen polynomialen Lösungsalgorithmus für dieses Problem gäbe, so wäre  $\mathbf{P}=\mathbf{NP}$ . In diesem Sinn sind die  $\mathbf{NP}$ -vollständigen Probleme die schwierigsten Probleme in  $\mathbf{NP}$ .

Viele kryptographische Protokolle sind so gemacht, dass die „guten“ Teilnehmer nur Probleme aus  $\mathbf{P}$  lösen müssen, während sich ein Angreifer vor Probleme aus  $\mathbf{NP}$  gestellt sieht.

Man weiß leider bis heute nicht, ob es Einwegfunktionen überhaupt gibt. Man kann aber zeigen, dass Einwegfunktionen genau dann existieren, wenn  $\mathbf{P} \neq \mathbf{NP}$  gilt [Balcazar1988, S.63].

Immer wieder behauptete jemand, er habe die Äquivalenz bewiesen (siehe [Hesselink2001]), aber bisher erwiesen sich diese Aussagen stets als falsch.

Es wurden eine Reihe von Algorithmen für Public-Key-Verfahren vorgeschlagen. Einige davon erwiesen sich, obwohl sie zunächst vielversprechend erschienen, als polynomial lösbar. Der berühmteste durchgefallene Bewerber ist der von Ralph Merkle [Merkle1978] vorgeschlagene Knapsack mit Falltür.

## 5.2 Knapsackproblem als Basis für Public-Key-Verfahren

### 5.2.1 Knapsackproblem

Gegeben  $n$  Gegenstände  $G_1, \dots, G_n$  mit den Gewichten  $g_1, \dots, g_n$  und den Werten  $w_1, \dots, w_n$ . Man soll wertmäßig so viel wie möglich unter Beachtung einer oberen Gewichtsschranke  $g$  davontragen. Gesucht ist also eine Teilmenge von  $\{G_1, \dots, G_n\}$ , etwa  $\{G_{i_1}, \dots, G_{i_k}\}$ , so dass  $w_{i_1} + \dots + w_{i_k}$  maximal wird unter der Bedingung  $g_{i_1} + \dots + g_{i_k} \leq g$ .

Derartige Fragen sind sogenannte  $\mathbf{NP}$ -vollständige Probleme (nicht deterministisch polynomial), die aufwendig zu berechnen sind.

Ein Spezialfall des Knapsackproblems ist:

Gegeben sind die natürlichen Zahlen  $a_1, \dots, a_n$  und  $g$ . Gesucht sind  $x_1, \dots, x_n \in \{0, 1\}$  mit  $g = \sum_{i=1}^n x_i a_i$  (wo also  $g_i = a_i = w_i$  gewählt ist). Dieses Problem heißt auch **0-1-Knapsackproblem** und wird mit  $K(a_1, \dots, a_n; g)$  bezeichnet.

Zwei 0-1-Knapsackprobleme  $K(a_1, \dots, a_n; g)$  und  $K(a'_1, \dots, a'_n; g')$  heißen kongruent, falls es zwei teilerfremde Zahlen  $w$  und  $m$  gibt, so dass

1.  $m > \max\{\sum_{i=1}^n a_i, \sum_{i=1}^n a'_i\}$ ,
2.  $g \equiv wg' \pmod{m}$ ,
3.  $a_i \equiv wa'_i \pmod{m}$  für alle  $i = 1, \dots, n$ .

#### Bemerkung:

Kongruente 0-1-Knapsackprobleme haben dieselben Lösungen. Ein schneller Algorithmus zur Klärung der Frage, ob zwei 0-1-Knapsackprobleme kongruent sind, ist nicht bekannt.

Das Lösen eines 0-1-Knapsackproblems kann durch Probieren der  $2^n$  Möglichkeiten für  $x_1, \dots, x_n$  erfolgen. Die beste Methode erfordert  $O(2^{n/2})$  Operationen, was für  $n = 100$  mit  $2^{100} \approx 1,27 \cdot 10^{30}$  und  $2^{n/2} \approx 1,13 \cdot 10^{15}$  für Computer eine unüberwindbare Hürde darstellt. Allerdings ist die Lösung für spezielle  $a_1, \dots, a_n$  recht einfach zu finden, etwa für  $a_i = 2^{i-1}$ . Die binäre Darstellung von  $g$  liefert unmittelbar  $x_1, \dots, x_n$ . Allgemein ist die Lösung des 0-1-Knapsackproblems leicht zu finden, falls eine Permutation<sup>3</sup>  $\pi$  von  $1, \dots, n$  mit  $a_{\pi(j)} > \sum_{i=1}^{j-1} a_{\pi(i)}$  existiert. Ist zusätzlich  $\pi$  die Identität, d.h.  $\pi(i) = i$  für  $i = 1, 2, \dots, n$ , so heißt die Folge  $a_1, \dots, a_n$  superwachsend. Das Verfahren 5.1 löst das Knapsackproblem mit superwachsender Folge im Zeitraum von  $O(n)$ .

---

**Krypto-Verfahren 5.1** Lösen von Knapsackproblemen mit superwachsenden Gewichten

---

```

for  $i = n$  to 1 do
  if  $T \geq a_i$  then
     $T := T - s_i$ 
     $x_i := 1$ 
  else
     $x_i := 0$ 
if  $T = 0$  then
   $X := (x_1, \dots, x_n)$  ist die Lösung.
else
  Es gibt keine Lösung.

```

---

### 5.2.2 Merkle-Hellman Knapsack-Verschlüsselung

1978 gaben Merkle und Hellman [Merkle1978] ein Public-Key-Verschlüsselungs-Verfahren an, das darauf beruht, das leichte 0-1-Knapsackproblem mit einer superwachsenden Folge in ein kongruentes mit einer nicht superwachsenden Folge zu „verfremden“. Es ist eine Blockchiffrierung, die bei jedem Durchgang einen  $n$  Bit langen Klartext chiffriert, siehe Krypto-Verfahren 5.2.

1982 gab Shamir [Shamir1982] einen Algorithmus zum Brechen des Systems in polynomialer Zeit an, ohne das allgemeine Knapsackproblem zu lösen. Len Adleman [Adleman1982] und Jeff Lagarias [Lagarias1983] gaben einen Algorithmus zum Brechen des 2-fachen iterierten Merkle-Hellman Knapsack-Verschlüsselungsverfahrens in polynomialer Zeit an. Ernst Brickell [Brickell1985] gab schließlich einen Algorithmus zum Brechen des mehrfachen iterierten Merkle-Hellman Knapsack-Verschlüsselungsverfahrens in polynomialer Zeit an. Damit war dieses Verfahren als Verschlüsselungsverfahren ungeeignet. Dieses Verfahren liefert also eine Einwegfunktion, deren Falltür-Information (Verfremden des 0-1-Knapsackproblems) durch einen Gegner entdeckt werden könnte.

---

<sup>3</sup>Eine Permutation  $\pi$  der Zahlen  $1, \dots, n$  ist die Vertauschung der Reihenfolge, in der diese Zahlen aufgezählt werden. Beispielsweise ist eine Permutation  $\pi$  von  $(1, 2, 3)$  gleich  $(3, 1, 2)$ , also  $\pi(1) = 3$ ,  $\pi(2) = 1$  und  $\pi(3) = 2$ .

---

**Krypto-Verfahren 5.2** Merkle-Hellman (auf Knapsackproblemen basierend)

---

Es sei  $(a_1, \dots, a_n)$  superwachsend. Seien  $m$  und  $w$  zwei teilerfremde Zahlen mit  $m > \sum_{i=1}^n a_i$  und  $1 \leq w \leq m-1$ . Wähle  $\bar{w}$  mit  $w\bar{w} \equiv 1 \pmod{m}$  die modulare Inverse von  $w$  und setze  $b_i := wa_i \pmod{m}$ ,  $0 \leq b_i < m$  für  $i = 1, \dots, n$ , und prüfe, ob die Folge  $b_1, \dots, b_n$  nicht superwachsend ist. Danach wird eine Permutation  $b_{\pi(1)}, \dots, b_{\pi(n)}$  von  $b_1, \dots, b_n$  publiziert und insgeheim die zu  $\pi$  inverse Permutation  $\mu$  festgehalten. Ein Sender schreibt seine Nachricht in Blöcke  $(x_1^{(j)}, \dots, x_n^{(j)})$  von Binärzahlen der Länge  $n$  und bildet

$$g^{(j)} := \sum_{i=1}^n x_i^{(j)} b_{\pi(i)}$$

und sendet  $g^{(j)}$ ,  $(j = 1, 2, \dots)$ .

Der Schlüsselinhaber bildet

$$G^{(j)} := \bar{w} g^{(j)} \pmod{m}, \quad 0 \leq G^{(j)} < m$$

und verschafft sich die  $x_{\mu(i)}^{(j)} \in \{0, 1\}$  (und somit auch die  $x_i^{(j)}$ ) aus

$$\begin{aligned} G^{(j)} &\equiv \bar{w} g^{(j)} = \sum_{i=1}^n x_i^{(j)} b_{\pi(i)} \bar{w} \equiv \sum_{i=1}^n x_i^{(j)} a_{\pi(i)} \pmod{m} \\ &= \sum_{i=1}^n x_{\mu(i)}^{(j)} a_{\pi(\mu(i))} = \sum_{i=1}^n x_{\mu(i)}^{(j)} a_i \pmod{m}, \end{aligned}$$

indem er die leichten 0-1-Knapsackprobleme  $K(a_1, \dots, a_n; G^{(j)})$  mit superwachsender Folge  $a_1, \dots, a_n$  löst.

---

## 5.3 Primfaktorzerlegung als Basis für Public-Key-Verfahren

### 5.3.1 Das RSA-Verfahren<sup>4,5</sup>

Bereits 1978 stellten R. Rivest, A. Shamir, L. Adleman [RSA1978] das bis heute wichtigste asymmetrische Kryptographie-Verfahren vor (Krypto-Verfahren 5.3).

**Bemerkung:**

Die Eulersche Phi-Funktion ist definiert durch:  $\phi(N)$  ist die Anzahl der zu  $N$  teilerfremden natürlichen Zahlen  $x \leq N$ . Zwei natürliche Zahlen  $a$  und  $b$  sind teilerfremd, falls  $\text{ggT}(a, b) = 1$ .

Für die Eulersche Phi-Funktion gilt:

$$\phi(1) = 1, \phi(2) = 1, \phi(3) = 2, \phi(4) = 2, \phi(6) = 2, \phi(10) = 4, \phi(15) = 8.$$

Zum Beispiel ist  $\phi(24) = 8$ , weil

$$|\{x < 24 : \text{ggT}(x, 24) = 1\}| = |\{1, 5, 7, 11, 13, 17, 19, 23\}|.$$

Ist  $p$  eine Primzahl, so gilt  $\phi(p) = p - 1$ .

---

<sup>4</sup>Vergleiche auch die Kapitel 4.10, ff.

<sup>5</sup>Mit CryptTool können Sie praktische Erfahrungen mit dem RSA-Verfahren sammeln: per Menü **Einzelverfahren** \ **RSA-Kryptosystem** \ **RSA-Demo**.

---

**Krypto-Verfahren 5.3** RSA (auf dem Faktorisierungsproblem basierend)

---

**Schlüsselgenerierung:**

Seien  $p$  und  $q$  zwei verschiedene Primzahlen und  $N = pq$ . Sei  $e$  eine frei wählbare, zu  $\phi(N)$  relative Primzahl, d.h.  $\text{ggT}(e, \phi(N)) = 1$ . Mit dem Euklidischen Algorithmus berechnet man die natürliche Zahl  $d < \phi(N)$ , so dass gilt

$$ed \equiv 1 \pmod{\phi(N)}.$$

Dabei ist  $\phi$  die **Eulersche Phi-Funktion**.

Der Ausgangstext wird in Blöcke zerlegt und verschlüsselt, wobei jeder Block einen binären Wert  $x^{(j)} \leq N$  hat.

**Öffentlicher Schlüssel:**

$$N, e.$$

**Privater Schlüssel:**

$$d.$$

**Verschlüsselung:**

$$y = e_T(x) = x^e \pmod{N}.$$

**Entschlüsselung:**

$$d_T(y) = y^d \pmod{N}$$

---

Kennt man die verschiedenen Primfaktoren  $p_1, \dots, p_k$  von  $N$ , so ist

$$\phi(N) = N \cdot \left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_k}\right).^6$$

Tabelle 5.1 zeigt die Werte bis 15. Im Spezialfall  $N = pq$  gilt

$$\phi(N) = pq(1 - 1/p)(1 - 1/q) = p(1 - 1/p)q(1 - 1/q) = (p - 1)(q - 1).$$

$n$	$\phi(n)$	Die zu $n$ teilerfremden natürlichen Zahlen kleiner $n$ .
1	1	1
2	1	1
3	2	1, 2
4	2	1, 3
5	4	1, 2, 3, 4
6	2	1, 5
7	6	1, 2, 3, 4, 5, 6
8	4	1, 3, 5, 7
9	6	1, 2, 4, 5, 7, 8
10	4	1, 3, 7, 9
15	8	1, 2, 4, 7, 8, 11, 13, 14

Tabelle 5.1: Eulersche Phi-Funktion

---

<sup>6</sup>Weitere Formeln zur Eulerschen Phi-Funktion finden sich in 4.8.2.

Die Funktion  $e_T$  ist eine Einwegfunktion, deren Falltür-Information die Primfaktorzerlegung von  $N$  ist.

Zur Zeit ist kein Algorithmus bekannt, der das Produkt zweier Primzahlen bei sehr großen Werten geeignet schnell zerlegen kann (z.B. bei mehreren hundert Dezimalstellen). Die heute schnellsten bekannten Algorithmen [Stinson1995] zerlegen eine zusammengesetzte ganze Zahl  $N$  in einem Zeitraum proportional zu  $L(N) = e^{\sqrt{\ln(N) \ln(\ln(N))}}$ . Einige Beispielwerte finden sich in Tabelle 5.2.

$N$	$10^{50}$	$10^{100}$	$10^{150}$	$10^{200}$	$10^{250}$	$10^{300}$
$L(N)$	$1,42 \cdot 10^{10}$	$2,34 \cdot 10^{15}$	$3,26 \cdot 10^{19}$	$1,20 \cdot 10^{23}$	$1,86 \cdot 10^{26}$	$1,53 \cdot 10^{29}$

Tabelle 5.2: Wertetabelle  $L(N)$

Bewiesen ist bis heute nicht, dass das Problem, RSA zu brechen, äquivalent zum Faktorisierungsproblem ist. Es ist aber klar, dass wenn das Faktorisierungsproblem „gelöst“ ist, dass dann das RSA-Verfahren nicht mehr sicher ist.<sup>7</sup>

### 5.3.2 Rabin-Public-Key-Verfahren (1979)

Für dieses Verfahren (5.4) konnte gezeigt werden, dass es äquivalent zum Brechen des Faktorisierungsproblems ist. Leider ist dieses Verfahren anfällig gegen Chosen-Ciphertext-Angriffe.

---

#### Krypto-Verfahren 5.4 Rabin (auf dem Faktorisierungsproblem basierend)

---

Seien  $p$  und  $q$  zwei verschiedene Primzahlen mit  $p, q \equiv 3 \pmod{4}$  und  $n = pq$ . Sei  $0 \leq B \leq n-1$ .

**Öffentlicher Schlüssel:**

$$e = (n, B).$$

**Privater Schlüssel:**

$$d = (p, q).$$

**Verschlüsselung:**

$$y = e_T(x) = x(x + B) \pmod{n}.$$

**Entschlüsselung:**

$$d_T(y) = \sqrt{y + B^2/4} - B/2 \pmod{n}.$$


---

Vorsicht: Wegen  $p, q \equiv 3 \pmod{4}$  ist die Verschlüsselung (mit Kenntnis des Schlüssels) leicht zu berechnen. Dies ist nicht der Fall für  $p \equiv 1 \pmod{4}$ . Außerdem ist die Verschlüsselungsfunktion nicht injektiv: Es gibt genau vier verschiedene Quellcodes, die  $e_T(x)$  als Urbild besitzen

---

<sup>7</sup>Im Jahre 2000 waren die Autoren der Ansicht, dass Werte der Größenordnung 100 bis 200 Dezimalstellen sicher sind. Sie schätzten, dass mit der aktuellen Computertechnik eine Zahl mit 100 Dezimalstellen bei vertretbaren Kosten in etwa zwei Wochen zerlegt werden könnte, dass mit einer teuren Konfiguration (z.B. im Bereich von 10 Millionen US-Dollar) eine Zahl mit 150 Dezimalstellen in etwa einem Jahr zerlegt werden könnte und dass eine 200-stellige Zahl noch für eine sehr lange Zeit unzerlegbar bleiben dürfte, falls es zu keinem mathematischen Durchbruch kommt. Dass es aber nicht doch schon morgen zu einem mathematischen Durchbruch kommt, kann man nie ausschließen.

Wie leicht man sich verschätzen kann, zeigt die Faktorisierung von RSA-200 (siehe Kapitel 4.11.4) – ganz ohne „mathematische Durchbrüche“.

$x, -x - B, \omega(x + B/2) - B/2, -\omega(x + B/2) - B/2$ , dabei ist  $\omega$  eine der vier Einheitswurzeln. Es muss also eine Redundanz der Quellcodes geben, damit die Entschlüsselung trotzdem eindeutig bleibt!

Hintertür-Information ist die Primfaktorzerlegung von  $n = pq$ .

## 5.4 Der diskrete Logarithmus als Basis für Public-Key-Verfahren<sup>8</sup>

Diskrete Logarithmen sind die Grundlage für eine große Anzahl von Algorithmen von Public-Key-Verfahren.

### 5.4.1 Der diskrete Logarithmus in $\mathbb{Z}_p^*$

Sei  $p$  eine Primzahl, und sei  $g$  ein Erzeuger der zyklischen multiplikativen Gruppe  $\mathbb{Z}_p^* = \{1, \dots, p-1\}$ . Dann ist die diskrete Exponentialfunktion zur Basis  $g$  definiert durch

$$e_g : k \longrightarrow y := g^k \mod p, \quad 1 \leq k \leq p-1.$$

Die Umkehrfunktion wird diskrete Logarithmusfunktion  $\log_g$  genannt; es gilt

$$\log_g(g^k) = k.$$

Unter dem Problem des diskreten Logarithmus (in  $\mathbb{Z}_p^*$ ) versteht man das folgende:

Gegeben  $p, g$  (ein Erzeuger der Gruppe  $\mathbb{Z}_p^*$ ) und  $y$ , bestimme  $k$  so, dass  $y = g^k \mod p$  gilt.

Die Berechnung des diskreten Logarithmus ist viel schwieriger als die Auswertung der diskreten Exponentialfunktion (siehe Kapitel 4.9). Tabelle 5.3 listet verschiedene Verfahren zur Berechnung des diskreten Logarithmus [Stinson1995] und ihre Komplexität.

Name	Komplexität
Babystep-Giantstep	$O(\sqrt{p})$
Silver-Pohlig-Hellman	polynomial in $q$ , dem größten Primteiler von $p-1$ .
Index-Calculus	$O(e^{(1+o(1))\sqrt{\ln(p)\ln(\ln(p))}})$

Tabelle 5.3: Verfahren zur Berechnung des diskreten Logarithmus in  $\mathbb{Z}_p^*$

Der aktuelle Rekord (Stand April 2007) für die Berechnung des diskreten Logarithmus wurde im Februar 2007 von der Gruppe Kleinjung, Franke und Bahr an der Universität Bonn aufgestellt.<sup>9</sup> Kleinjung berechnete den diskreten Logarithmus modulo einer 160-stelligen Prim-

<sup>8</sup>In dem Lernprogramm **ZT** können Sie mit der Verteilung des diskreten Logarithmus experimentieren und Shanks Babystep-Giantstep-Methode anwenden: Siehe Lern-Kapitel 6.1-6.3, Seiten 1-6/6.

ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen. Siehe Anhang A.4.

<sup>9</sup>[http://www.nabble.com/Discrete-logarithms-in-GF\(p\)-----160-digits-t3175622.html](http://www.nabble.com/Discrete-logarithms-in-GF(p)-----160-digits-t3175622.html)

zahl  $p$  und Erzeuger  $g$ :

$$\begin{aligned} p &= \lfloor 10^{159} \pi \rfloor + 119849 \\ &= 314159265358979323846264338327950288419716939937510582097494 \\ &\quad 459230781640628620899862803482534211706798214808651328230664 \\ &\quad 7093844609550582231725359408128481237299 \\ g &= 2 \end{aligned}$$

Konkret wurde der diskrete Logarithmus  $k$  von folgender Zahl  $y$  berechnet:<sup>10</sup>

$$\begin{aligned} y &= \lfloor 10^{159} e \rfloor \\ &= 271828182845904523536028747135266249775724709369995957496696 \\ &\quad 762772407663035354759457138217852516642742746639193200305992 \\ &\quad 1817413596629043572900334295260595630738 \\ k &= \log_g(y) \mod p \\ &= 829897164650348970518646802640757844024961469323126472198531 \\ &\quad 845186895984026448342666252850466126881437617381653942624307 \\ &\quad 537679319636711561053526082423513665596 \end{aligned}$$

Die Suche wurde mit der GNFS-Methode (General Number Field Sieve, Index-Calculus) durchgeführt und benötigte ca. 17 CPU-Jahre auf 3.2 GHz Xeon Maschinen.

#### 5.4.2 Diffie-Hellman-Schlüsselvereinbarung<sup>11</sup>

Die Mechanismen und Algorithmen der klassischen Kryptographie greifen erst dann, wenn die Teilnehmer bereits den geheimen Schlüssel ausgetauscht haben. Im Rahmen der klassischen Kryptographie führt kein Weg daran vorbei, dass Geheimnisse kryptographisch ungesichert ausgetauscht werden müssen. Die Sicherheit der Übertragung muss hier durch nicht-kryptographische Methoden erreicht werden. Man sagt dazu, dass man zum Austausch der Geheimnisse einen geheimen Kanal braucht; dieser kann physikalisch oder organisatorisch realisiert sein.

Das Revolutionäre der modernen Kryptographie ist unter anderem, dass man keine geheimen Kanäle mehr braucht: Man kann geheime Schlüssel über nicht-geheime, also öffentliche Kanäle vereinbaren.

Ein Protokoll, das dieses Problem löst, ist das von Diffie und Hellman (Krypto-Verfahren 5.5).

---

##### **Krypto-Verfahren 5.5** Diffie-Hellman-Schlüsselvereinbarung

---

Zwei Teilnehmer  $A$  und  $B$  wollen einen gemeinsamen geheimen Schlüssel vereinbaren.

Sei  $p$  eine Primzahl und  $g$  eine natürliche Zahl. Diese beide Zahlen müssen nicht geheim sein. Zunächst wählen sich die beiden Teilnehmer je eine geheime Zahl  $a$  bzw.  $b$ . Daraus bilden sie die Werte  $\alpha = g^a \mod p$  und  $\beta = g^b \mod p$ . Dann werden die Zahlen  $\alpha$  und  $\beta$  ausgetauscht. Schließlich potenziert jeder den erhaltenen Wert mit seiner geheimen Zahl und erhält  $\beta^a \mod p$  bzw.  $\alpha^b \mod p$ .

Damit gilt

$$\beta^a \equiv (g^b)^a \equiv g^{ba} \equiv g^{ab} \equiv (g^a)^b \equiv \alpha^b \mod p$$


---

Die Sicherheit des **Diffie-Hellman-Protokolls** hängt eng mit der Berechnung der diskreten Logarithmus modulo  $p$  zusammen. Es wird sogar vermutet, dass diese Probleme äquivalent sind.

<sup>10</sup>Die Zahl  $y$  ergab sich aus den ersten 159 Stellen der Eulerschen Zahl  $e$ .

<sup>11</sup>In CrypTool ist dieses Austauschprotokoll visualisiert: Sie können die einzelnen Schritte mit konkreten Zahlen nachvollziehen per Menü **Einzelverfahren \ Protokolle \ Diffie-Hellman-Demo**.



### 5.4.3 ElGamal-Public-Key-Verschlüsselungsverfahren in $\mathbb{Z}_p^*$

Indem man das Diffie-Hellman Schlüsselvereinbarungsprotokoll leicht variiert, kann man einen asymmetrischen Verschlüsselungsalgorithmus erhalten (Krypto-Verfahren 5.6). Diese Beobachtung geht auf Taher ElGamal zurück.

---

**Krypto-Verfahren 5.6** ElGamal (auf dem diskreten Logarithmusproblem basierend)

---

Sei  $p$  eine Primzahl, so dass der diskrete Logarithmus in  $\mathbb{Z}_p^*$  schwierig zu berechnen ist. Sei  $\alpha \in \mathbb{Z}_p^*$  ein primitives Element. Sei  $a \in \mathbb{N}$  eine natürliche Zahl und  $\beta = \alpha^a \mod p$ .

**Öffentlicher Schlüssel:**

$$p, \alpha, \beta.$$

**Privater Schlüssel:**

$$a.$$

Sei  $k \in \mathbb{Z}_{p-1}$  eine zufällige Zahl und  $x \in \mathbb{Z}_p^*$  der Klartext.

**Verschlüsselung:**

$$e_T(x, k) = (y_1, y_2),$$

wobei

$$y_1 = \alpha^k \mod p,$$

und

$$y_2 = x\beta^k \mod p.$$

**Entschlüsselung:**

$$d_T(y_1, y_2) = y_2(y_1^a)^{-1} \mod p.$$

---

### 5.4.4 Verallgemeinertes ElGamal-Public-Key-Verschlüsselungsverfahren

Den diskreten Logarithmus kann man in beliebigen endlichen Gruppen  $(G, \circ)$  verallgemeinern. Im Folgenden geben wir einige Eigenschaften über die Gruppe  $G$  an, damit das diskrete Logarithmusproblem schwierig wird.

**Berechnung der diskreten Exponentialfunktion** Sei  $G$  eine Gruppe mit der Operation  $\circ$  und  $g \in G$ . Die (diskrete) Exponentialfunktion zur Basis  $g$  ist definiert durch

$$e_g : k \mapsto g^k, \quad \text{für alle } k \in \mathbb{N}.$$

Dabei definiert man

$$g^k := \underbrace{g \circ \dots \circ g}_{k \text{ mal}}.$$

Die Exponentialfunktion ist leicht zu berechnen:

**Lemma**

*Die Potenz  $g^k$  kann in höchstens  $2 \log_2 k$  Gruppenoperationen berechnet werden.*

**Beweis**

Sei  $k = 2^n + k_{n-1}2^{n-1} + \dots + k_12 + k_0$  die Binärdarstellung von  $k$ . Dann ist  $n \leq \log_2(k)$ , denn

$2^n \leq k < 2^{n+1}$ .  $k$  kann in der Form  $k = 2k' + k_0$  mit  $k' = 2^{n-1} + k_{n-1}2^{n-2} + \dots + k_1$  geschrieben werden. Es folgt

$$g^k = g^{2k' + k_0} = (g^{k'})^2 g^{k_0}.$$

Man erhält also  $g^k$  aus  $g^{k'}$  indem man einmal quadriert und eventuell mit  $g$  multipliziert. Damit folgt die Behauptung durch Induktion nach  $n$ .  $\square$

### Problem des diskreten Logarithmus'

Sei  $G$  eine endliche Gruppe mit der Operation  $\circ$ . Sei  $\alpha \in G$  und  $\beta \in H = \{\alpha^i : i \geq 0\}$ .  
 Gesucht ist der eindeutige  $a \in \mathbb{N}$  mit  $0 \leq a \leq |H| - 1$  und  $\beta = \alpha^a$ .  
 Wir bezeichnen  $a$  mit  $\log_\alpha(\beta)$ .

**Berechnung des diskreten Logarithmus'** Ein einfaches Verfahren zur Berechnung des diskreten Logarithmus' eines Gruppenelements, das wesentlich effizienter ist als das bloße Durchprobieren aller möglichen Werte für  $k$ , ist der Babystep-Giantstep-Algorithmus.

**Satz 5.4.1** (Babystep-Giantstep-Algorithmus). *Sei  $G$  eine Gruppe und  $g \in G$ . Sei  $n$  die kleinste natürliche Zahl mit  $|G| \leq n^2$ . Dann kann der diskrete Logarithmus eines Elements  $h \in G$  zur Basis  $g$  berechnet werden, indem man zwei Listen mit jeweils  $n$  Elementen erzeugt und diese Listen vergleicht.*

*Zur Berechnung dieser Listen braucht man  $2n$  Gruppenoperationen.*

### Beweis

Zuerst bilde man die zwei Listen

Giantstep-Liste:  $\{1, g^n, g^{2n}, \dots, g^{n \cdot n}\},$

Babystep-Liste:  $\{hg^{-1}, hg^{-2}, \dots, hg^{-n}\}.$

Falls  $g^{jn} = hg^{-i}$ , also  $h = g^{i+jn}$ , so ist das Problem gelöst. Falls die Listen disjunkt sind, so ist  $h$  nicht als  $g^{i+jn}, i, j \leq n$ , darstellbar. Da dadurch alle Potenzen von  $g$  erfasst werden, hat das Logarithmusproblem keine Lösung.  $\square$

Man kann sich mit Hilfe des Babystep-Giantstep-Algorithmus klar machen, dass die Berechnung des diskreten Logarithmus' sehr viel schwieriger ist als die Berechnung der diskreten Exponentialfunktion. Wenn die auftretenden Zahlen etwa 1000 Bit Länge haben, so benötigt man zur Berechnung von  $g^k$  nur etwa 2000 Multiplikationen, zur Berechnung des diskreten Logarithmus' mit dem Babystep-Giantstep-Algorithmus aber etwa  $2^{500} \approx 10^{150}$  Operationen.

Neben dem Babystep-Giantstep-Algorithmus gibt es noch zahlreiche andere Verfahren zur Berechnung des diskreten Logarithmus' [Stinson1995].

**Der Satz von Silver-Pohlig-Hellman** In endlichen abelschen Gruppen lässt sich das diskrete Logarithmusproblem in Gruppen kleinerer Ordnung reduzieren.

**Satz 5.4.2** (Silver-Pohlig-Hellman). *Sei  $G$  eine endliche abelsche Gruppe mit  $|G| = p_1^{a_1} p_2^{a_2} \cdot \dots \cdot p_s^{a_s}$ . Dann lässt sich das diskrete Logarithmusproblem in  $G$  auf das Lösen von Logarithmusproblemen in Gruppen der Ordnung  $p_1, \dots, p_s$  zurückführen.*

Enthält  $|G|$  einen „dominanten“ Primteiler  $p$ , so ist die Komplexität des Logarithmusproblems ungefähr

$$O(\sqrt{p}).$$

Wenn also das Logarithmusproblem schwer sein soll, so muss die Ordnung der verwendeten Gruppe  $G$  einen großen Primteiler haben. Insbesondere gilt, wenn die diskrete Exponentialfunktion in der Gruppe  $\mathbb{Z}_p^*$  eine Einwegfunktion sein soll, so muss  $p - 1$  einen großen Primteiler haben. In diesem Fall kann man ein verallgemeinertes ElGamal-Verfahren definieren (Krypto-Verfahren 5.7).

---

**Krypto-Verfahren 5.7** Verallgemeinertes ElGamal (auf dem diskreten Logarithmusproblem basierend)

---

Sei  $G$  eine endliche Gruppe mit Operation  $\circ$ , und sei  $\alpha \in G$ , so dass der diskrete Logarithmus in  $H = \{\alpha^i : i \geq 0\}$  schwer ist. Sei  $a$  mit  $0 \leq a \leq |H| - 1$  und sei  $\beta = \alpha^a$ .

**Öffentlicher Schlüssel:**

$$\alpha, \beta.$$

**Privater Schlüssel:**

$$a.$$

Sei  $k \in \mathbb{Z}_{|H|}$  eine zufällige Zahl und  $x \in G$  ein Klartext.

**Verschlüsselung:**

$$e_T(x, k) = (y_1, y_2),$$

wobei

$$y_1 = \alpha^k,$$

und

$$y_2 = x \circ \beta^k.$$

**Entschlüsselung:**

$$d_T(y_1, y_2) = y_2 \circ (y_1^a)^{-1}.$$


---

Elliptische Kurven liefern nützliche Gruppen für Public-Key-Verschlüsselungsverfahren.

# Literaturverzeichnis

- [Adleman1982] Adleman L.:  
*On breaking the iterated Merkle-Hellman public key Cryptosystem.*  
Advances in Cryptologie, Proceedings of Crypto 82, Plenum Press 1983, 303-308.
- [Balcazar1988] Balcazar J.L., Daaz J., Gabarr J.:  
*Structural Complexity I.*  
Springer Verlag, pp 63.
- [Brickell1985] Brickell E.F.:  
*Breaking Iterated Knapsacks.*  
Advances in Cryptology: Proc. CRYPTO'84, Lecture Notes in Computer Science, vol. 196,  
Springer Verlag, New York, 1985, pp. 342-358.
- [Hesselink2001] Hesselink Wim H.:  
*The borderline between P and NP.*  
<http://www.cs.rug.nl/~wim/pub/whh237.pdf>,  
Februar 12, 2001.
- [Lagarias1983] Lagarias J.C.:  
*Knapsack public key Cryptosystems and diophantine Approximation.*  
Advances in Cryptology, Proceedings of Crypto 83, Plenum Press.
- [Merkle1978] Merkle R. and Hellman M.:  
*Hiding information and signatures in trapdoor knapsacks.*  
IEEE Trans. Information Theory, IT-24, 1978.
- [RSA1978] Rivest R.L., Shamir A. and Adleman L.:  
*A Method for Obtaining Digital Signatures and Public Key Cryptosystems.*  
Commun. ACM, vol 21, April 1978, pp. 120-126.
- [Shamir1982] Shamir A.:  
*A polynomial time algorithm for breaking the basic Merkle-Hellman Cryptosystem.*  
Symposium on Foundations of Computer Science (1982), 145-152.
- [Stinson1995] Stinson D.R.:  
*Cryptography.*  
CRC Press, Boca Raton, London, Tokyo, 1995.

## Kapitel 6

# Hashfunktionen und Digitale Signaturen

(Schneider J. / Esslinger B. / Koy H., Juni 2002; Updates: Feb. 2003, Juni 2005, Juli 2009)

Ziel der digitalen Signatur ist es, folgende zwei Punkte zu gewährleisten:

- Benutzerauthentizität:  
Es kann überprüft werden, ob eine Nachricht tatsächlich von einer bestimmten Person stammt.
- Nachrichtenintegrität:  
Es kann überprüft werden, ob die Nachricht (unterwegs) verändert wurde.

Zum Einsatz kommt wieder eine asymmetrische Technik (siehe Verschlüsselungsverfahren). Ein Teilnehmer, der eine digitale Signatur für ein Dokument erzeugen will, muss ein Schlüsselpaar besitzen. Er benutzt seinen geheimen Schlüssel, um Signaturen zu erzeugen, und der Empfänger benutzt den öffentlichen Schlüssel des Absenders, um die Richtigkeit der Signatur zu überprüfen. Es darf wiederum nicht möglich sein, aus dem öffentlichen den geheimen Schlüssel abzuleiten<sup>1</sup>.

Im Detail sieht ein *Signaturverfahren* folgendermaßen aus:

Der Absender berechnet aus seiner Nachricht und seinem geheimen Schlüssel die digitale Signatur der Nachricht. Im Vergleich zur handschriftlichen Unterschrift hat die digitale Signatur den Vorteil, dass die Unterschrift auch vom unterschriebenen Dokument abhängt. Die Unterschriften ein und desselben Teilnehmers sind verschieden, sofern die unterzeichneten Dokumente nicht vollkommen übereinstimmen. Selbst das Einfügen eines Leerzeichens in den Text würde zu einer anderen Signatur führen. Eine Verletzung der Nachrichtenintegrität wird also vom Empfänger der Nachricht erkannt, da in diesem Falle die Signatur nicht mehr zum Dokument passt und sich bei der Überprüfung als unkorrekt erweist.

Das Dokument wird samt Signatur an den Empfänger verschickt. Dieser kann mit Hilfe des öffentlichen Schlüssels des Absenders, des Dokuments und der Signatur feststellen, ob die Signatur korrekt ist. Das gerade beschriebene Verfahren hätte in der Praxis jedoch einen

---

<sup>1</sup>Mit CrypTool können Sie ebenfalls digitale Signaturen erzeugen und prüfen: in den Untermenüs des Hauptmenüpunktes **Digitale Signaturen / PKI** oder per **Einzelverfahren \ RSA-Kryptosystem \ Signaturdemo (Signaturerzeugung)**.

entscheidenden Nachteil: Die Signatur wäre ungefähr genauso lang wie das eigentliche Dokument. Um den Datenverkehr nicht unnötig anwachsen zu lassen und aus Performance-Gründen wendet man – vor dem Signieren – auf das Dokument eine kryptographische Hashfunktion<sup>2</sup> an. Deren Output wird dann signiert.

*Stanislaw Lem*<sup>3</sup>:

Wir können alles aus dieser Welt machen, nur nicht eine Welt, in der die Menschen in einigen zigtausend Jahren überlegen könnten: 'So, es ist nun genug. So soll es von nun an für immer bleiben. Verändern wir nichts, erfinden wir nichts, weil es besser nicht sein kann, und wenn doch, dann wollen wir es nicht.'

## 6.1 Hashfunktionen

Eine *Hashfunktion*<sup>4</sup> bildet eine Nachricht beliebiger Länge auf eine Zeichenfolge mit konstanter Größe, den Hashwert, ab.

### 6.1.1 Anforderungen an Hashfunktionen

Kryptographisch sichere Hashfunktionen erfüllen folgende drei Anforderungen (Reihenfolge so, dass die Anforderungen ansteigen):

- Standhaftigkeit gegen 1st-Pre-Image-Attacks:  
Es sollte praktisch unmöglich sein, zu einer gegebenen Zahl eine Nachricht zu finden, die genau diese Zahl als Hashwert hat.  
Gegeben (fix): Hashwert  $H'$ ,  
Gesucht: Nachricht  $m$ , so dass gilt:  $H(m) = H'$ .
- Standhaftigkeit gegen 2nd-Pre-Image-Attacks:  
Es sollte praktisch unmöglich sein, zu einer gegebenen Nachricht eine zweite Nachricht zu

---

<sup>2</sup>Hashfunktionen sind in CrypTool an mehreren Stellen implementiert.

In den Menüs **Einzelverfahren** \ **Hashverfahren** bzw. **Analyse** \ **Hashverfahren** haben Sie die Möglichkeit

- 6 Hashfunktionen auf den Inhalt des aktiven Fensters anzuwenden,
- für eine Datei den Hashwert zu berechnen,
- in der Hash-Demo die Auswirkung von Textänderungen auf den Hashwert zu testen,
- aus einem Passwort gemäß dem PKCS#5-Standard einen Schlüssel zu berechnen,
- aus einem Text und einem geheimen Schlüssel HMACs zu berechnen, und
- aufgrund von gezielt gesuchten Hashwertkollisionen einen Angriff auf digitale Signaturen zu simulieren.

<sup>3</sup>Antwort von Stanislaw Lem auf die Kritik an seinem philosophischen Hauptwerk „Summa Technologiae“, 1964, in der er die evolutionäre Möglichkeit einer Entstehung der künstlichen Intelligenz ausführte.

<sup>4</sup>Hashverfahren berechnen eine komprimierte Repräsentation elektronischer Daten (Message). Die Verarbeitung dieser Message durch das Hashverfahren ergibt als Output einen sogenannten Message Digest. Message Digests sind typischerweise zwischen 128 und 512 Bit lang – abhängig vom Algorithmus. Sichere Hashverfahren werden typischerweise mit anderen kryptographischen Algorithmen kombiniert, wie z.B. Digitale-Signatur-Algorithmen, Keyed-Hash Message Authentication Codes, oder bei der Erzeugung von Zufallszahlen (Bits) benutzt.

finden, die genau denselben Hashwert hat.

Gegeben (fix): Nachricht  $m_1$  [und damit der Hashwert  $H_1 = H(m_1)$ ],

Gesucht: Nachricht  $m_2$ , so dass gilt:  $H(m_2) = H_1$ .

- Standhaftigkeit gegen Kollisionsangriffe:

Es sollte es praktisch unmöglich sein, zwei (beliebige) Nachrichten mit demselben Hashwert (welcher ist egal) zu finden.

Gesucht: 2 Nachrichten  $m_1$  und  $m_2$ , so dass gilt:  $H(m_1) = H(m_2)$ .

### 6.1.2 Aktuelle Angriffe gegen Hashfunktionen wie SHA-1

Bisher konnte die Existenz von perfekt sicheren kryptographischen Hashfunktionen nicht formal bewiesen werden.

Über mehrere Jahre gab es keine neuen Attacks gegen Hashverfahren, und allgemein wurde den Kandidaten, die in der Praxis bislang keine Schwächen in ihrer Struktur gezeigt hatten (zum Beispiel SHA-1<sup>5</sup> oder RIPEMD-160<sup>6</sup>) vertraut.

Auf der Crypto 2004 (August 2004)<sup>7</sup> wurde dieses Sicherheitsgefühl jedoch stark in Zweifel gezogen: Chinesische Wissenschaftler veröffentlichten Kollisionsangriffe gegen MD4, SHA-0 und Teile von SHA-1, die weltweit zu einer starken Beschäftigung mit neuen Hash-Angriffen führte.

Die zunächst veröffentlichten Resultate reduzierten den erwarteten Aufwand für die Suche nach einer SHA-1 Kollision von  $2^{80}$  (brute-force) auf  $2^{69}$  [Wang2005]. In der Folge wurden Verfahren angekündigt, die den Aufwand weiter auf  $2^{63}$  [Wang2005b] und  $2^{52}$  [McDonald2009] reduzieren sollen. Damit wäre der Kollisionsangriff in den Bereich des praktisch möglichen gerückt, denn ähnliche Aufwände wurden in der Vergangenheit schon realisiert (s. 1.1.2).

Die Sicherheit bereits erstellter Signaturen wird durch den geschilderten Kollisionsangriff aber nicht gefährdet.

Nach dem aktuellen Kenntnisstand ist keine Panik angesagt, aber für digitale Signaturen sollten zumindest in Zukunft längere Hashwerte und/oder andere Verfahren benutzt werden.

Das U.S. National Institute of Standards and Technology (NIST) hat schon vor Bekanntwerden der neuen Ergebnisse angekündigt, SHA-1 in den nächsten Jahren auslaufen zu lassen. Es ist daher zu empfehlen, für neue Produkte zur Erstellung von Signaturen SHA-1 nicht mehr zu

<sup>5</sup>SHA-1 ist eine in den Standards FIPS 180-1 (durch die US-Behörde NIST), ANSI X9.30 Part 2 und [FIPS186] spezifizierte 160-Bit Hashfunktion.

SHA bedeutet "Secure Hash Algorithm" und wird häufig benutzt, z.B. mit DSA, RSA oder ECDSA.

Der aktuelle Standard [FIPS180-3] definiert vier sichere Hashverfahren – SHA-1, SHA-256, SHA-384 und SHA-512. Für diese Hashalgorithmen sind in der Testsuite FIPS 140-2 auch Validierungstests definiert.

Die Ausgabelänge der SHA-Algorithmen wurde vergrößert aufgrund der Möglichkeit von Geburtstagsangriffen: diese machen – grob gesprochen – den  $n$ -Bit AES und ein  $2n$ -bit Hashverfahren äquivalent:

- 128-bit AES – SHA-256

- 192-bit AES – SHA-384

- 256-bit AES – SHA-512.

Mit CrypTool können Sie den Geburtstagsangriff auf digitale Signaturen nachvollziehen:

über das Menü **Analyse \ Hashverfahren \ Angriff auf den Hashwert der digitalen Signatur**.

<sup>6</sup>RIPEMD-160, RIPEMD-128 und die optionale Erweiterung RIPEMD-256 haben Object Identifier, definiert von der ISO-identifizierten Organisation TeleTrusT, sowohl für Hashverfahren als auch in Kombination mit RSA. RIPEMD-160 ist Teil des internationalen ISO/IEC-Standards ISO/IEC 10118-3:1998 für dedizierte Hashfunktionen, zusammen mit RIPEMD-128 und SHA-1. Weitere Details:

- <http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html>

- <http://www.ietf.org/rfc/rfc2857.txt> ("The Use of HMAC-RIPEMD-160-96 within ESP and AH").

<sup>7</sup><http://www.iacr.org/conferences/crypto2004/>

verwenden. Die SHA-2 Familie [FIPS180-3] bietet stärkere Verfahren. Um den neuen Erkenntnissen in der Kryptoanalyse von Hashfunktionen Rechnung zu tragen, hat das NIST 2008 einen Wettbewerb gestartet, in dem eine neue Hashfunktion "SHA-3" ausgewählt werden soll.<sup>8</sup> Der Wettbewerb soll im Jahr 2012 enden.

Weitere Informationen zu diesem Thema finden sich z.B. in dem Artikel „Hash mich – Konsequenzen der erfolgreichen Angriffe auf SHA-1“ von Reinhard Wobst und Jürgen Schmidt<sup>9</sup> von Heise Security.

### 6.1.3 Signieren mit Hilfe von Hashfunktionen

Das Signatur-Verfahren mit Hashfunktion (ohne den oben genannten Nachteil) sieht folgendermaßen aus:<sup>10</sup>

Anstatt das eigentliche Dokument zu signieren, berechnet der Absender nun zuerst den Hashwert der Nachricht und signiert diesen. Der Empfänger bildet ebenfalls den Hashwert der Nachricht (der benutzte Algorithmus muss bekannt sein). Er überprüft dann, ob die mitgeschickte Signatur eine korrekte Signatur des Hashwertes ist. Ist dies der Fall, so wurde die Signatur korrekt verifiziert. Die Authentizität der Nachricht ist damit gegeben, da wir angenommen hatten, dass aus der Kenntnis des öffentlichen Schlüssels nicht der geheime Schlüssel abgeleitet werden kann. Dieser geheime Schlüssel wäre jedoch notwendig, um Nachrichten in einem fremden Namen zu signieren.

Einige digitale Signaturverfahren basieren auf asymmetrischer Verschlüsselung, das bekannteste Beispiel dieser Gattung ist RSA. Für die RSA-Signatur verwendet man die gleiche mathematische Operation wie zum Entschlüsseln, nur wird sie auf den Hash-Wert des zu unterschreibenden Dokuments angewendet.

Andere Systeme der digitalen Signatur wurden, wie DSA (Digital Signature Algorithm), ausschließlich zu diesem Zweck entwickelt, und stehen in keiner direkten Verbindung zu einem entsprechenden Verschlüsselungsverfahren.

Beide Signaturverfahren, RSA und DSA, werden in den folgenden beiden Abschnitten näher beleuchtet. Anschließend gehen wir einen Schritt weiter und zeigen, wie basierend auf der elektronischen Unterschrift das digitale Pendant zum Personalausweis entwickelt wurde. Dieses Verfahren nennt man Public-Key-Zertifizierung.

## 6.2 RSA-Signatur

Wie im Kommentar am Ende von Abschnitt 4.10.3 bemerkt, ist es möglich, die RSA-Operationen mit dem privaten und öffentlichen Schlüssel in umgekehrter Reihenfolge auszuführen, d. h.  $M \text{ hoch } d \text{ hoch } e \pmod{N}$  ergibt wieder  $M$ . Wegen dieser simplen Tatsache ist es möglich, RSA als Signaturverfahren zu verwenden.

Eine RSA-Signatur  $S$  zur die Nachricht  $M$  wird durch folgende Operation mit dem privaten

---

<sup>8</sup><http://csrc.nist.gov/groups/ST/hash/sha-3/>

<sup>9</sup><http://www.heise.de/security/artikel/56555>.

Weitere Quellen sind z.B.:

<http://www.bsi.bund.de/esig/basics/techbas/krypto/index.htm>

<http://csrc.nist.gov/CryptoToolkit/tkhash.html>.

<sup>10</sup>Vergleiche auch:

[http://de.wikipedia.org/wiki/Digitale\\_Signatur](http://de.wikipedia.org/wiki/Digitale_Signatur),

[http://en.wikipedia.org/wiki/Digital\\_signature](http://en.wikipedia.org/wiki/Digital_signature).



Schlüssel erzeugt:

$$S \equiv M^d \pmod{N}$$

Zur Verifikation wird die korrespondierende Public-Key-Operation auf der Signatur  $S$  ausgeführt und das Ergebnis mit der Nachricht  $M$  verglichen:

$$S^e \equiv (M^d)^e \equiv (M^e)^d \equiv M \pmod{N}$$

Wenn das Ergebnis  $S^e$  mit der Nachricht  $M$  übereinstimmt, dann akzeptiert der Prüfer die Signatur, andernfalls ist die Nachricht entweder verändert worden, oder sie wurde nicht vom Inhaber von  $d$  unterschrieben.

Wie weiter oben erklärt, werden Signaturen in der Praxis nie direkt auf der Nachricht ausführt, sondern auf einem kryptographischen Hashwert davon. Um verschiedene Attacken auf das Signaturverfahren (und seine Kombination mit Verschlüsselung) auszuschließen, ist es nötig, den Hashwert vor der Exponentiation auf spezielle Weise zu formatieren, wie in PKCS#1 (Public Key Cryptography Standard #1 [PKCS1]) beschrieben. Der Tatsache, dass dieser Standard nach mehreren Jahren Einsatz revidiert werden musste, kann als Beispiel dafür dienen, wie schwer es ist, kryptographische Details richtig zu definieren.

## 6.3 DSA-Signatur

Im August 1991 hat das U.S. National Institute of Standards and Technology (NIST) einen digitalen Signaturalgorithmus (DSA, Digital Signature Algorithm) vorgestellt, der später zum U.S. Federal Information Processing Standard (FIPS 186 [FIPS186]) wurde.

Der Algorithmus ist eine Variante des ElGamal-Verfahrens. Seine Sicherheit beruht auf dem Diskreten Logarithmus Problem. Die Bestandteile des privaten und öffentlichen DSA-Schlüssels, sowie die Verfahren zur Signatur und Verifikation sind in Krypto-Verfahren 6.1 zusammengefasst.

Obwohl DSA unabhängig von einem Verschlüsselungsverfahren so spezifiziert wurde, dass es aus Länder exportiert werden kann, die den Export von kryptographischer Hard- und Software einschränken (wie die USA zum Zeitpunkt der Spezifikation), wurde festgestellt [Schneier1996, S. 490], dass die Operationen des DSA dazu geeignet sind, nach RSA bzw. ElGamal zu verschlüsseln.

## 6.4 Public-Key-Zertifizierung

Ziel der Public-Key-Zertifizierung ist es, die Bindung eines öffentlichen Schlüssels an einen Benutzers zu garantieren und nach außen nachvollziehbar zu machen. In Fällen, in denen nicht sichergestellt werden kann, dass ein öffentlicher Schlüssel auch wirklich zu einer bestimmten Person gehört, sind viele Protokolle nicht mehr sicher, selbst wenn die einzelnen kryptographischen Bausteine nicht geknackt werden können.

### 6.4.1 Die Impersonalisierungsattacke

Angenommen Charlie hat zwei Schlüsselpaare (PK1, SK1) und (PK2, SK2). Hierbei bezeichnet SK den geheimen Schlüssel (secret key) und PK den öffentlichen Schlüssel (public key). Weiter angenommen, es gelingt ihm, Alice PK1 als öffentlichen Schlüssel von Bob und Bob PK2 als

---

## Krypto-Verfahren 6.1 DSA-Signatur

---

### Öffentlicher Schlüssel

$p$  prim

$q$  160-Bit Primfaktor von  $p - 1$

$g = h^{(p-1)/q} \bmod p$ , wobei  $h < p - 1$  und  $h^{(p-1)/q} > 1 \pmod{p}$

$y \equiv g^x \bmod p$

**Bemerkung:** Die Parameter  $p, q$  und  $g$  können von einer Gruppe von Benutzern gemeinsam genutzt werden.

### Privater Schlüssel

$x < q$  (160-Bit Zahl)

### Signatur

$m$  zu signierende Nachricht

$k$  zufällig gewählte Primzahl, kleiner als  $q$

$r = (g^k \bmod p) \bmod q$

$s = (k^{-1}(\text{SHA-1}(m) + xr)) \bmod q$

### Bemerkung:

- $(s, r)$  ist die Signatur.
- Die Sicherheit der Signatur hängt nicht nur von der Mathematik ab, sondern auch von der Verfügbarkeit einer guten Zufallsquelle für  $k$ .
- SHA-1 ist eine 160-Bit Hashfunktion.

### Verifikation

$w = s^{-1} \bmod q$

$u_1 = (\text{SHA-1}(m)w) \bmod q$

$u_2 = (rw) \bmod q$

$v = (g^{u_1}y^{u_2}) \bmod p \bmod q$

**Bemerkung:** Wenn  $v = r$ , dann ist die Signatur gültig.

---

öffentlichen Schlüssel von Alice „unterzujubeln“ (etwa indem er ein öffentliches Schlüsselverzeichnis fälscht).

Dann ist folgender Angriff möglich:

- Alice möchte eine Nachricht an Bob senden. Sie verschlüsselt diese mit PK1, da sie denkt, dies sei Bobs öffentlicher Schlüssel. Anschließend signiert sie die Nachricht mit ihrem geheimen Schlüssel und schickt sie ab.
- Charlie fängt die Nachricht ab, entfernt die Signatur und entschlüsselt die Nachricht mit SK1. Wenn er möchte, kann er die Nachricht anschließend nach Belieben verändern. Dann verschlüsselt er sie wieder, aber diesmal mit dem echten öffentlichen Schlüssel von Bob, den er sich aus einem öffentlichen Schlüsselverzeichnis geholt hat, signiert sie mit SK2 und schickt die Nachricht weiter an Bob.
- Bob überprüft die Signatur mit PK2 und wird zu dem Ergebnis kommen, dass die Signatur in Ordnung ist. Dann entschlüsselt er die Nachricht mit seinem geheimen Schlüssel.

Charlie ist so in der Lage, die Kommunikation zwischen Alice und Bob abzuhören und die ausgetauschten Nachrichten zu verändern, ohne dass dies von den beteiligten Personen bemerkt wird. Der Angriff funktioniert auch, wenn Charlie nur ein Schlüsselpaar hat.

Ein anderer Name für diese Art von Angriffen ist „Man-in-the-Middle-Attack“. Hilfe gegen diese Art von Angriffen verspricht die Public-Key-Zertifizierung, die die Authentizität öffentlicher Schlüssel garantieren kann. Die am weitesten verbreitete Zertifizierungsmethode ist der X.509-Standard.

#### 6.4.2 X.509-Zertifikat

Jeder Teilnehmer, der sich per X.509-Zertifikat [X.509] die Zugehörigkeit seines öffentlichen Schlüssels zu seiner realen Person bestätigen lassen möchte, wendet sich an eine sogenannte Certification Authority (CA)<sup>11</sup>. Dieser beweist er seine Identität (etwa durch Vorlage seines Personalausweises). Anschließend stellt die CA ihm ein elektronisches Dokument (Zertifikat) aus, in dem im wesentlichen der Name des Zertifikatnehmers und der Name der CA, der öffentliche Schlüssel des Zertifikatnehmers und der Gültigkeitszeitraum des Zertifikats vermerkt sind. Die CA unterzeichnet das Zertifikat anschließend mit ihrem geheimen Schlüssel.

Jeder kann nun anhand des öffentlichen Schlüssels der CA überprüfen, ob das Zertifikat unverfälscht ist. Die CA garantiert also die Zugehörigkeit von Benutzer und öffentlichem Schlüssel.

Dieses Verfahren ist nur so lange sicher, wie die Richtigkeit des öffentlichen Schlüssels der CA sichergestellt ist. Aus diesem Grund lässt jede CA ihren öffentlichen Schlüssel bei einer anderen CA zertifizieren, die in der Hierarchie über ihr steht. In der obersten Hierarchieebene (Wurzelinstanz) gibt es in der Regel nur eine CA, die dann natürlich keine Möglichkeit mehr hat, sich ihren Schlüssel bei einer anderen CA zertifizieren zu lassen. Sie ist also darauf angewiesen, ihren Schlüssel auf andere Art und Weise gesichert zu übermitteln. Bei vielen Software-Produkten, die mit Zertifikaten arbeiten (zum Beispiel den Webbrowsern von Microsoft und Netscape) sind die Zertifikate dieser Wurzel-CAs schon von Anfang an fest in das Programm eingebettet und können auch vom Benutzer nachträglich nicht mehr geändert werden. Aber auch durch öffentliche Bekanntgabe in Zeitungen können (öffentliche) CA-Schlüssel gesichert übermittelt werden.

---

<sup>11</sup> Oft auch Trustcenter oder im deutschen Signaturgesetz „Zertifizierungsdiensteanbieter“ genannt, wenn die Zertifikate nicht nur einer geschlossenen Benutzergruppe angeboten werden.

# Literaturverzeichnis

- [FIPS180-3] U.S. Department of Commerce/N.I.S.T. ,  
*Secure Hash Standard (SHS)*,  
October 2008.  
(FIPS 180-3 supersedes FIPS 180-2.)
- [FIPS186] U.S. Department of Commerce/N.I.S.T. ,  
*Entity authentication using public key cryptography*,  
Februar 18, 1997.  
Nicht mehr gültig.
- [FIPS186-2] U.S. Department of Commerce/N.I.S.T. ,  
*Digital Signature Standard (DSS)*,  
Januar 27, 2000. Change Note: Oktober 5, 2001.  
<http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf>
- [McDonald2009] Cameron McDonald, Philip Hawkes, Josef Pieprzyk,  
*Differential Path for SHA-1 with complexity  $O(2^{52})$* ,  
<http://eprint.iacr.org/2009/259>
- [PKCS1] RSA Laboratories,  
*PKCS #1 v2.1 Draft 3: RSA Cryptography Standard*,  
April 19, 2002.
- [Schneier1996] Bruce Schneier,  
*Applied Cryptography, Protocols, Algorithms, and Source Code in C*,  
Wiley, 2nd edition, 1996.
- [Wang2005] Xiaoyun Wang, Yiqun Yin, Hongbo Yu,  
*Finding Collisions in the Full SHA-1*,  
Advances in Cryptology-Crypto 2005, LNCS 3621: 17-36, 2005.
- [Wang2005b] Xiaoyun Wang, Andrew Yao and Frances Yao,  
*New Collision Search for SHA-1*,  
Crypto 2005 Rump Session  
<http://www.iacr.org/conferences/crypto2005/rumpSchedule.html>
- [Wobst2005] Reinhard Wobst,  
*New Attacks Against Hash Functions*,  
Information Security Bulletin, April 2005.
- [X.509] ITU-T,  
*ITU-T Recommendation X.509 (1997 E): Information Technology – Open Systems In-*

*terconnection – The Directory: Authentication Framework*,  
Juni 1997.

- [X.509v3] ITU-T,  
*X.509 (1993) Amendment 1: Certificate Extensions, The Directory Authentication Framework*,  
International Telecommunication Union, Geneva, Switzerland, July 1995  
(equivalent to amendment 1 to ISO/IEC 9594-8).

# Kapitel 7

## Elliptische Kurven

(Filipovics B. / Büger M. / Esslinger B. / Oyono R., April 2000, Updates: Dez. 2001, Juni 2002, März 2003, November 2009)

### 7.1 Elliptische Kurven – ein effizienter Ersatz für RSA?

Bei Datenübertragungen kommt es auf Sicherheit und Effizienz an. Viele Anwendungen verwenden den RSA-Algorithmus als asymmetrisches Signatur- und Verschlüsselungsverfahren.

Solange die verwendeten Schlüssel hinreichend lang sind, bestehen keinerlei Bedenken gegen die *Sicherheit* des RSA-Verfahrens. Allerdings hat die Entwicklung der Rechnerleistungen in der vergangenen Jahren dazu geführt, dass die benötigten Schlüssellängen mehrfach angehoben werden mussten (vergleiche Kapitel 4.11). Da die meisten Chips auf Smartcards nicht in der Lage sind, längere Schlüssel als 1024 Bit zu verarbeiten, besteht Bedarf für Alternativen zum RSA. Elliptische Kurven können eine solche Alternative bieten.

Die *Effizienz* eines kryptographischen Algorithmus hängt wesentlich von der benötigten Schlüssellänge und vom Rechenaufwand ab, um ein vorgeschriebenes Sicherheitsniveau zu erreichen. Der entscheidende Vorteil Elliptischer Kurven im Vergleich zum RSA-Algorithmus liegt in der Tatsache, dass die sicheren Schlüssellängen erheblich kürzer sind.

Setzt man den Trend, dass sich die Leistung der verfügbaren Rechner im Schnitt alle 18 Monate verdoppelt (Gesetz von Moore<sup>1</sup>), in die Zukunft fort, so kann man von einer Entwicklung der sicheren Schlüssellängen wie in Abbildung 7.1 ausgehen [Lenstra1999] (Quelle: Arjen Lenstra und Eric Verheul: <http://cryptosavvy.com/table.htm>).

---

<sup>1</sup>empirische Erkenntnis von Gordon Moore, Mitbegründer von Intel, 1965.

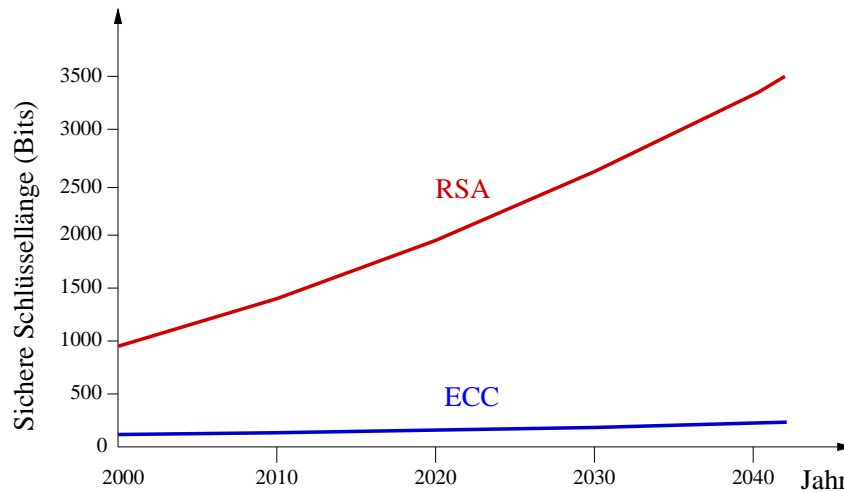


Abbildung 7.1: Prognose für die Entwicklung der als sicher betrachteten Schlüssellängen für RSA und Elliptische Kurven

Bei der digitalen Signatur muss man differenzieren: für die *Erstellung* einer digitalen Signatur benötigen auf Elliptischen Kurven basierende Verfahren im Vergleich zu RSA nur gut ein Zehntel des Rechenaufwandes (66 zu 515 Ganzzahlmultiplikationen). Siehe hierzu Abbildung 7.2 (Quelle: Dr. J. Merkle, Elliptic Curve Cryptography Workshop, 2001). Betrachtet man die für eine *Verifikation* durchzuführenden Rechenschritte, dreht sich dieses Bild jedoch zu Gunsten von RSA um (112 zu 17 Ganzzahlmultiplikationen). Der Grund liegt darin, dass es bei Verwendung des RSA möglich ist, einen sehr kurzen Exponent für den öffentlichen Schlüssel zu wählen, solange der private Exponent nur hinreichend lang ist.

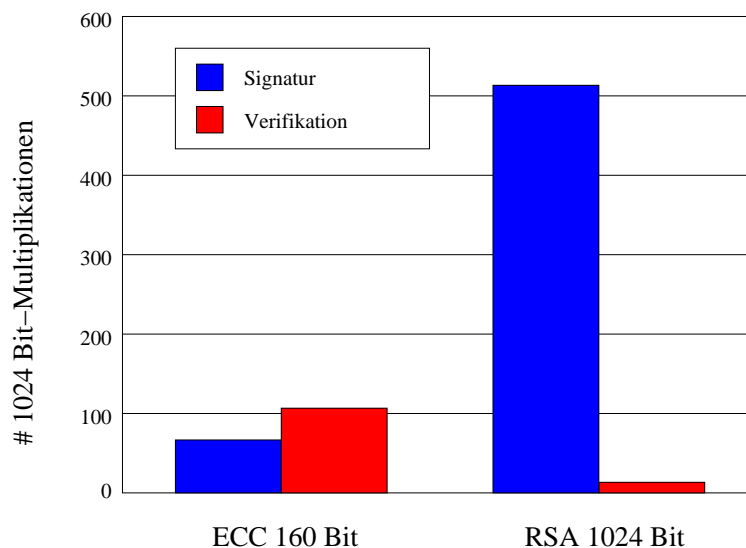


Abbildung 7.2: Gegenüberstellung des Aufwands der Operationen Signieren und Verifizieren bei RSA und Elliptischen Kurven

Da bei Smartcards, die auf RSA basieren, stets der lange (private) Schlüssel auf der Karte gespeichert werden muss und die Erstellung der digitalen Signatur, nicht aber die Verifikation, auf der Karte stattfindet, treten hier deutlich die Vorteile Elliptischer Kurven zutage.

Das größte Problem bei der Implementierung von Verfahren, die auf Elliptischen Kurven beruhen, ist bislang die mangelnde *Standardisierung*. Es gibt nur eine RSA-Implementierung, aber viele Arten, Elliptische Kurven einzusetzen. So können verschiedene Zahlkörper zugrunde gelegt, eine Vielzahl von (Elliptischen) Kurven — durch Parameter beschrieben<sup>2</sup> — eingesetzt und unterschiedliche Darstellungen der Kurvenpunkte verwendet werden. Jede Wahl hat ihre Vorzüge, so dass für jede Anwendung eine andere Implementierung optimal sein kann. Dies hat jedoch zur Konsequenz, dass Systeme, die auf Elliptischen Kurven beruhen, oftmals nicht interoperabel sind. Um mit einer beliebigen auf Elliptischen Kurven basierenden Anwendung kommunizieren zu können, müsste man eine Vielzahl von Implementierungen vorhalten, was den Effizienzvorteil gegenüber der Verwendung von RSA zunichte macht.

Deshalb bemühen sich internationale Organisationen um Standardisierung: IEEE (P1363), ASC (ANSI X9.62, X9.63), ISO/IEC sowie RSA Laboratories und Certicom. Im Gegensatz zur IEEE, die bisher nur eine Beschreibung der verschiedenen Implementierungen vorgenommen hat, hat die ASC konkret 10 Kurven ausgewählt und empfiehlt deren Verwendung. Der Vorteil des ASC-Ansatzes ist, dass ein einziges Byte ausreicht, um die verwendete Kurve zu spezifizieren. Zur Zeit ist jedoch nicht absehbar, ob es der ASC gelingen wird, einen de-facto-Standard durchzusetzen.

Obwohl aktuell kein Handlungsbedarf besteht<sup>3</sup>, laufende RSA-Anwendungen umzustellen, sollte man bei der Neuimplementierung ernsthaft den Einsatz von Verfahren erwägen, die auf Elliptischen Kurven basieren. Dies gilt insbesondere, wenn es sich um Anwendungen im Finanzsektor handelt, die noch nach 2005<sup>4</sup> operativ sein sollen.

## 7.2 Elliptische Kurven – Historisches

Auf dem Gebiet der Elliptischen Kurven wird seit über 100 Jahren geforscht. Im Laufe der Zeit hat man viele weitläufige und mathematisch tiefgründige Resultate im Zusammenhang mit Elliptischen Kurven gefunden und veröffentlicht. Ein Mathematiker würde sagen, dass die Elliptischen Kurven (bzw. die dahinterstehende Mathematik) gut verstanden sind. Ursprünglich war diese Forschung reine Mathematik, das heißt Elliptische Kurven wurden zum Beispiel in den mathematischen Teilgebieten Zahlentheorie und algebraische Geometrie untersucht, die allgemein sehr abstrakt sind. Auch in der nahen Vergangenheit spielten Elliptische Kurven eine bedeutende Rolle in der reinen Mathematik. In den Jahren 1993 und 1994 veröffentlichte Andrew Wiles mathematische Arbeiten, die weit über das Fachpublikum hinaus auf große Begeisterung gestoßen sind. In diesen Arbeiten bewies er die Richtigkeit einer — in den sechziger Jahren des 20. Jahrhunderts von zwei Japanern aufgestellten — Vermutung. Dabei geht es kurz und grob gesagt um den Zusammenhang zwischen Elliptischen Kurven und sogenannten Modulformen. Das für die meisten eigentlich Interessante daran ist, dass Wiles mit seinen Arbeiten auch den berühmten zweiten Satz von Fermat bewiesen hat. Dieser Satz hatte sich seit Jahrhunderten (Fermat lebte von 1601 bis 1665) einem umfassenden Beweis durch die Mathematik entzogen. Dementsprechend groß war die Resonanz auf den Beweis durch Wiles. In der Formulierung von Fermat lautet der nach ihm benannte Satz so (Fermat hat folgende Worte an den Rand des 1621 von Bachet herausgegebenen Werks *Diophants* geschrieben):

*Cubum autem in duos cubos, aut quadratoquadratum in duos quadratoquadratos, et generaliter nullam in infinitum ultra quadratum potestatem in duos ejusdem nominis*

<sup>2</sup>Siehe Kapitel 7.4

<sup>3</sup>Aktuelle Informationen zur Sicherheit des RSA-Verfahrens finden Sie in Kapitel 4.11.

<sup>4</sup>BSI-Empfehlung: "Geeignete Kryptoalgorithmen" vom 24. Oktober 2002.



*fas est dividere: cujus rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.*

Frei übersetzt und mit der Schreibweise der heutigen Mathematik bedeutet dies:

Es gibt keine positiven ganzen Zahlen  $x, y$  und  $z$  größer als Null, so dass  $x^n + y^n = z^n$  für  $n > 2$  gilt. Ich habe einen bemerkenswerten Beweis für diese Tatsache gefunden, aber es ist nicht genug Platz am Rand [des Buches], um ihn niederzuschreiben.

Dies ist schon bemerkenswert: Eine relativ einfach zu verstehende Aussage (gemeint ist Fermats zweiter Satz) konnte erst nach so langer Zeit bewiesen werden, obwohl Fermat selber angab, schon einen Beweis gefunden zu haben. Im übrigen ist der Beweis von Wiles sehr umfangreich (alle im Zusammenhang mit dem Beweis stehenden Veröffentlichungen von Wiles ergeben schon ein eigenes Buch). Man sollte sich daher im klaren sein, dass die Elliptischen Kurven im allgemeinen sehr tiefgreifende Mathematik berühren.

Soweit zur Rolle der Elliptischen Kurven in der reinen Mathematik. Im Jahr 1985 haben Neal Koblitz und Victor Miller unabhängig voneinander vorgeschlagen, Elliptische Kurven in der Kryptographie einzusetzen. Damit haben die Elliptischen Kurven auch eine ganz konkrete praktische Anwendung gefunden. Ein weiteres interessantes Einsatzgebiet für Elliptische Kurven ist die Faktorisierung von ganzen Zahlen (auf der Schwierigkeit/Komplexität, die Primfaktoren einer sehr großen Zahl zu finden, beruht das RSA-Kryptosystem: vergleiche Kapitel 4.11 ). In diesem Bereich werden seit 1987 Verfahren untersucht und eingesetzt, die auf Elliptischen Kurven basieren (vergleiche Kapitel 7.8).

Es gibt auch Primzahltests, die auf Elliptischen Kurven basieren.

Elliptische Kurven werden in den verschiedenen Gebieten unterschiedlich eingesetzt. Kryptographische Verfahren auf Basis von Elliptischen Kurven beruhen auf der Schwierigkeit eines als Elliptische Kurven Diskreter Logarithmus bekannten Problems.

Ferner gibt es einen Zusammenhang zwischen Faktorisierung von ganzen Zahlen und Elliptischen Kurven, der in Abschnitt 7.8 näher beschrieben wird.

## 7.3 Elliptische Kurven – Mathematische Grundlagen

In diesem Abschnitt erhalten Sie Informationen über *Gruppen* und *Körper*.

### 7.3.1 Gruppen

Da der Begriff der *Gruppe* umgangssprachlich anders als in der Mathematik eingesetzt wird, soll der Vollständigkeit halber an dieser Stelle die wesentliche Aussage der formalen Definition einer Gruppe kurz eingeführt werden:

- Eine Gruppe ist eine nichtleere Menge  $G$  mit einer Verknüpfung “ $\cdot$ ”. Die Menge  $G$  ist unter der Verknüpfung  $\cdot$  abgeschlossen, d.h., sind  $a, b$  Elemente aus  $G$ , so ist auch ihre Verknüpfung  $ab = a \cdot b$  ein Element aus  $G$ .
- Für alle Elemente  $a, b$  und  $c$  aus  $G$  gilt:  $(ab)c = a(bc)$  (Assoziativgesetz).
- Es gibt ein Element  $e$  in  $G$ , das sich bezüglich der Verknüpfung  $\cdot$  neutral verhält, d.h., für alle  $a$  aus der Menge  $G$  gilt  $ae = ea = a$ .

- Zu jedem Element  $a$  aus  $G$  gibt es ein *inverses Element*<sup>5</sup>  $a^{-1}$  (in  $G$ ), so dass gilt:  $aa^{-1} = a^{-1}a = e$ .

Gilt zusätzlich noch für alle  $a, b$  aus  $G$ , dass  $ab = ba$  (Kommutativgesetz), so nennt man die Gruppe  $G$  eine *abelsche* Gruppe.

Da man auf der selben Menge mehrere Verknüpfung erklären kann, unterscheidet man diese durch verschiedene Namensgebungen und Zeichen (z.B.  $+$  Addition oder  $\cdot$  Multiplikation).

Als einfachstes Beispiel einer (abelschen) Gruppe sei die Gruppe der ganzen Zahlen mit der üblichen Addition genannt. Die Menge der ganzen Zahlen wird mit  $\mathbb{Z}$  bezeichnet.  $\mathbb{Z}$  hat unendlich viele Elemente, denn  $\mathbb{Z} = \{\dots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \dots\}$ . Die Verknüpfung von zum Beispiel  $1 + 2$  liegt in  $\mathbb{Z}$ , denn  $1 + 2 = 3$  und  $3$  liegt in  $\mathbb{Z}$ . Das neutrale Element der Gruppe  $\mathbb{Z}$  ist  $0$ . Das Inverse Element von  $3$  ist  $-3$ , denn  $3 + (-3) = 0$ .

Für unsere Zwecke besonders interessant sind sogenannte *endliche* Gruppen, bei denen die zugrundelegte Menge  $G$  nur aus einer endlichen Anzahl von Elementen besteht. Beispiele sind die Gruppen  $\mathbb{Z}_n = \{0, 1, 2, 3, \dots, n-1\}$ ,  $n$  der Teilerreste bei der Division durch  $n$  mit der Addition als Verknüpfung.

**Zyklische Gruppen** Als *zyklische Gruppen*<sup>6</sup> bezeichnet man solche Gruppen  $G'$ , die ein Element  $g$  besitzen, aus dem man mittels der Gruppen-Verknüpfung alle anderen Elemente der Gruppe erzeugen kann. Es gibt also für jedes Element  $a$  aus  $G'$  eine positive, ganze Zahl  $i$ , so dass die  $i$ -fache Verknüpfung von  $g$  mit sich selbst  $g^i = g \cdot g \cdots g = a$ . Das Element  $g$  ist ein *Generator* der zyklischen Gruppe — jedes Element in  $G'$  lässt sich mittels  $g$  und der Verknüpfung erzeugen.

**Ordnung von Elementen einer Gruppe** Nun zur Ordnung eines Elements der Gruppe: Sei  $a$  aus  $G$ . Die kleinste positive ganze Zahl  $r$  für die gilt, dass  $a^r$ , also  $r$  mal  $a$  mit sich selbst verknüpft, das neutrale Element der Gruppe  $G'$  ist (d.h.  $a^r = e$ ), nennt man *Ordnung* von  $a$ .

Die *Ordnung der Gruppe* ist die Anzahl der Elemente in der Menge  $G$ . Ist die Gruppe  $G$  zyklisch und  $g$  ein Generator, so stimmt die Ordnung von  $g$  mit der Gruppenordnung überein. Man kann leicht zeigen, dass die Ordnung eines Gruppenelements stets die Gruppenordnung teilt. Hieraus folgt insbesondere, dass Gruppen mit Primzahlordnung (d.h. die Ordnung der Gruppe ist eine Primzahl) zyklisch sind.

## 7.3.2 Körper

In praktischen Anwendungen betrachtet man häufig Mengen, auf denen nicht nur eine (Gruppen-) Verknüpfung, sondern zwei Verknüpfungen definiert sind. Diese nennt man oft Addition und Multiplikation. Die mathematisch interessantesten Mengen dieser Art sind sogenannte Körper, wie z.B. die Menge der reellen Zahlen.

Unter einem Körper versteht man in der Mathematik eine Menge  $K$  mit den zwei Verknüpfungen Addition und Multiplikation (mit  $+$  und  $\cdot$  bezeichnet), so dass die folgenden Bedingungen erfüllt sind:

<sup>5</sup>Das inverse Element ist eindeutig bestimmt, denn sind  $x, y \in G$  zwei Inverse zu  $a$ , d.h. gilt  $ax = xa = e$  und  $ay = ya = e$ , so folgt  $x = xe = x(ay) = (xa)y = ey = y$ .

<sup>6</sup>Zyklische Gruppen können grundsätzlich auch unendlich sein wie z.B. die additive Gruppe der ganzen Zahlen. Wir betrachten hier jedoch nur endliche zyklische Gruppen.

- Die Menge  $K$  ist zusammen mit der Verknüpfung  $+$  (Addition) eine abelsche Gruppe. Dabei sei  $0$  das neutrale Element der Verknüpfung  $+$ .
- Die Menge  $K \setminus \{0\}$  (d.h.  $K$  ohne das Element  $0$ ) ist zusammen mit der Verknüpfung  $\cdot$  (Multiplikation) ebenfalls eine abelsche Gruppe. Dabei sei  $1$  das neutrale Element der Verknüpfung  $\cdot$ .
- Für alle Elemente  $a, b$  und  $c$  aus  $K$  gilt  $c \cdot (a + b) = c \cdot a + c \cdot b$  und  $(a + b) \cdot c = a \cdot c + b \cdot c$  (Distributivgesetz).

Körper können endlich viele oder unendliche viele Elemente enthalten — je nach dem nennt man den Körper *endlich* oder *unendlich*. So sind die uns vertrauten Körper der rationalen bzw. der reellen Zahlen unendlich. Beispiele für endliche Körper sind die Primkörper  $\mathbb{Z}_p = \{0, 1, 2, 3, \dots, p-1\}$ ,  $p$  eine Primzahl, versehen mit der Addition modulo  $p$  und der Multiplikation modulo  $p$  (auch Restklassenkörper genannt).

**Charakteristik eines Körpers** Die Charakteristik eines Körper  $K$  ist die Ordnung des neutralen Elements der Multiplikation (1-Element) bezüglich der Addition, d.h. die kleinste natürliche Zahl  $n$ , so dass gilt

$$\underbrace{1 + 1 + \dots + 1}_{n \text{ mal}} = 0,$$

wobei  $0$  das neutrale Element der Addition ist. Gibt es keine solche natürliche Zahl, d.h. ergibt  $1 + 1 + \dots + 1$  unabhängig von der Zahl der Summanden nie das neutrale Element der Addition  $0$ , so sagt man, der Körper habe Charakteristik  $0$ .

Körper mit Charakteristik  $0$  haben daher stets die (paarweise verschiedenen) Elemente  $1, 1+1, 1+1+1, \dots$  und sind folglich stets unendlich; andererseits können Körper mit endlicher Charakteristik durchaus endlich oder auch unendlich sein. Ist die Charakteristik  $n$  endlich, so muss sie eine Primzahl sein, denn wäre sie zusammengesetzt, d.h.  $n = pq$ , so sind  $p, q < n$  und aufgrund der Minimalität der Charakteristik ist keines der Körperelemente  $\bar{p} = \underbrace{1 + 1 + \dots + 1}_{p \text{ mal}}$ ,

$\bar{q} = \underbrace{1 + 1 + \dots + 1}_{q \text{ mal}}$  gleich  $0$ . Folglich existieren Inverse  $\bar{p}^{-1}, \bar{q}^{-1}$  bezüglich der Multiplikation.

Dann ist aber  $(\bar{p}\bar{q})(\bar{p}^{-1}\bar{q}^{-1}) = 1$ , andererseits ist nach Definition der Charakteristik  $\bar{p}\bar{q} = \bar{n} = \underbrace{1 + 1 + \dots + 1}_{n \text{ times}} = 0$  und somit  $\underbrace{(\bar{p}\bar{q})}_{=0}(\bar{p}^{-1}\bar{q}^{-1}) = 0$ , was zu einem Widerspruch führt.

**Beispiel:** Der Körper  $\mathbb{Z}_p$ ,  $p$  prim, hat die Charakteristik  $p$ . Ist  $p$  nicht prim, so ist  $\mathbb{Z}_p$  gar kein Körper.

Der einfachste, denkbare Körper ist  $\mathbb{Z}_2 = \{0, 1\}$ , der nur Null- und Einselement enthält. Dabei ist  $0 + 0 = 0$ ,  $0 + 1 = 1 + 0 = 1$ ,  $1 + 1 = 0$ ,  $1 \cdot 1 = 1$ ,  $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$ .

**Endliche Körper** Wie bereits erwähnt, hat jeder endliche Körper eine Charakteristik  $p \neq 0$ , wobei  $p$  eine Primzahl ist. Zu jeder Primzahl  $p$  gibt es einen Körper mit  $p$  Elementen, nämlich  $\mathbb{Z}_p$ .

Die Anzahl der Elemente eines Körpers muss jedoch im allgemeinen keine Primzahl sein. So ist es nicht schwer, einen Körper mit 4 Elementen zu konstruieren<sup>7</sup>.

<sup>7</sup>Die Menge  $K = \{0, 1, a, b\}$  ist mit den Verknüpfungen der folgenden Tabellen ein Körper:

Man kann zeigen, dass die Ordnung jedes Körpers eine Primzahlpotenz (d.h. die Potenz einer Primzahl) ist. Andererseits kann man zu jeder Primzahlpotenz  $p^n$  einen Körper konstruieren, der die Ordnung  $p^n$  hat. Da zwei endliche Körper mit gleicher Zahl von Elementen nicht unterscheidbar<sup>8</sup> sind, spricht man von **dem Körper mit  $p^n$  Elementen** und bezeichnet diesen mit  $GF(p^n)$ . Dabei steht  $GF$  für *Galois Feld* in Erinnerung an den französischen Mathematiker Galois.

Eine besondere Rolle spielen die Körper  $GF(p)$ , deren Ordnung eine Primzahl ist. Man nennt solche Körper Primkörper zur Primzahl  $p$  und bezeichnet ihn meist mit  $\mathbb{Z}_p$ <sup>9</sup>.

## 7.4 Elliptische Kurven in der Kryptographie

In der Kryptographie betrachten wir Elliptische Kurven. Solche Kurven ergeben sich als Lösungen einer Gleichung der Form<sup>10</sup>

$$F(x_1, x_2, x_3) = -x_1^3 + x_2^2 x_3 + a_1 x_1 x_2 x_3 - a_2 x_1^2 x_3 + a_3 x_2 x_3^2 - a_4 x_1 x_3^2 - a_6 x_3^3 = 0. \quad (7.1)$$

Dabei sind die Variablen  $x_1, x_2, x_3$  sowie die Parameter  $a_1, \dots, a_4, a_6$  Elemente eines gegebenen Körpers  $K$ . Körper und Parameter müssen so gewählt werden, dass die Kurve bestimmte, für die Kryptographie relevante Eigenschaften besitzt. Der zugrunde liegende Körper  $K$  kann einfach die bekannte Menge der reellen Zahlen oder auch ein endlicher Körper sein (vgl. letzter Abschnitt). Damit sich eine sinnvolle Kurve ergibt, müssen die Parameter so gewählt sein, dass die folgenden Nebenbedingungen gelten

$$\frac{\partial F}{\partial x_1} \neq 0, \quad \frac{\partial F}{\partial x_2} \neq 0, \quad \frac{\partial F}{\partial x_3} \neq 0.$$

Ferner betrachten wir Punkte, die sich nur durch eine Vervielfachung jeder Komponente ergeben, als identisch, denn mit  $(x_1, x_2, x_3)$  erfüllt stets auch  $\alpha(x_1, x_2, x_3)$  die Ausgangsgleichung. Formal betrachten wir daher Äquivalenzklassen von Punkten  $(x_1, x_2, x_3)$ , wobei wir zwei Punkte als gleich ansehen, wenn sie durch Multiplikation mit einer Konstante  $\alpha$  auseinander hervorgehen. Setzt man in der Ausgangsgleichung  $x_3 = 0$ , so wird diese zu  $-x_1^3 = 0$ , also  $x_1 = 0$ . Folglich ist die Äquivalenzklasse, die das Element  $(0, 1, 0)$  enthält, die einzige Punkt mit  $x_3 = 0$ . Für alle

+	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

und

·	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

<sup>8</sup>Sind  $K, K'$  zwei Körper mit  $k = p^n$  Elementen, so gibt es eine eindeutige Abbildung  $\varphi : K \rightarrow K'$ , die sich mit der Körperarithmetik verträgt. Eine solche Abbildung nennt man Isomorphie. Isomorphe Körper verhalten sich mathematisch gleich, so dass es keinen Sinn macht, zwischen ihnen zu unterscheiden. Z.B. sind  $\mathbb{Z}_2$  und  $K' = \{NULL, EINS\}$  mit Nullelement  $NULL$  und Einselement  $EINS$  isomorph. Hierbei sei darauf hingewiesen, dass mathematische Objekte ausschließlich über ihre Eigenschaften definiert sind.

<sup>9</sup>Für Primkörper sind die additive Gruppe sowie die multiplikative Gruppe zyklisch. Ferner enthält jeder Körper  $GF(p^n)$  einen zu  $\mathbb{Z}_p$  isomorphen Primkörper.

<sup>10</sup>Die hier verwendete Kurve erhält man als Nullstellen des *Polynoms*  $F$  vom Grad drei in drei Variablen. Dabei bezeichnet man allgemein Ausdrücke der Form  $P = \sum_{i_1, \dots, i_n \in \mathbb{N}_0} a_{i_1 \dots i_n} x_1^{i_1} \dots x_n^{i_n}$  mit Koeffizienten  $a_{i_1 \dots i_n} \in K$  als Polynome in  $n$  Variablen  $x_1, \dots, x_n$  über dem Körper  $K$ , wenn  $\text{grad } P := \max\{i_1 + \dots + i_n : a_{i_1 \dots i_n} \neq 0\}$  einen endlichen Wert hat, die Summe also nur aus endlich vielen Summanden (Monomen) besteht. Die Summe der Exponenten der Variablen jedes einzelnen Summanden ist maximal 3, bei mindestens einem Summanden kommt 3 als Exponentwert einer Variablen auch wirklich vor.

anderen Lösungspunkte können wir die Transformation

$$K \times K \times (K \setminus \{0\}) \ni (x_1, x_2, x_3) \mapsto (x, y) := \left( \frac{x_1}{x_3}, \frac{x_2}{x_3} \right) \in K \times K$$

vornehmen, die die Anzahl der Variablen von drei auf zwei reduziert. Die Ausgangsgleichung  $F(x_1, x_2, x_3) = 0$  war so gewählt, dass sich auf diese Weise die sogenannte Weierstrass-Gleichung<sup>11</sup>

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (7.2)$$

ergibt. Da alle bis auf einen Lösungspunkt durch die Gleichung (7.2) beschrieben werden können, bezeichnet man (7.2) auch oft als die Elliptische Gleichung, ihre Lösungsmenge folglich mit

$$\mathbf{E} = \{(x, y) \in K \times K \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\}.$$

Dabei soll  $\mathcal{O}$  den auf diese Weise nicht beschriebenen Punkt  $(0, 1, 0)$  darstellen, der durch die Projektion (Division durch  $x_3$ ) quasi in den unendlich fernen Punkt abgebildet wird.

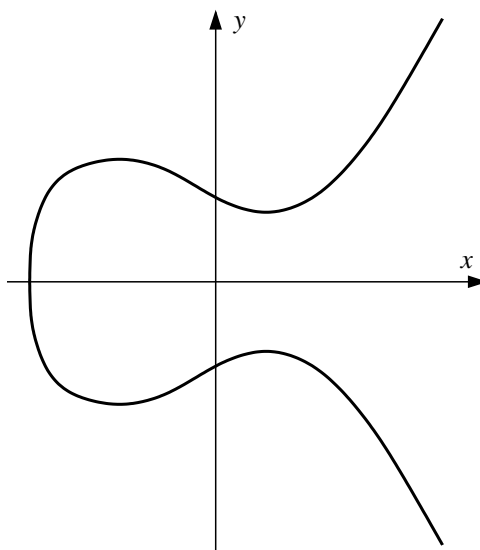


Abbildung 7.3: Beispiel einer Elliptischen Kurve über dem Körper der reellen Zahlen.

Als zugrunde liegenden Körper für eine Elliptische Kurve verwendet man in der Kryptographie stets endliche Körper  $K = GF(p^n)$ , also nicht wie in Abbildung 7.3 die zu einer stetigen Kurve führenden reellen Zahlen. Der Grund liegt, einfach gesagt, darin, dass wir bei der Verarbeitung und Übertragung von Nachrichten stets nur endlich viele Zustände zur Verfügung haben (aufgrund der Arbeitsweise moderner Computer), Körper mit unendlich vielen Elementen wie z.B. die reellen Zahlen daher stets nur unvollständig darstellen können.

In der Praxis hat es sich als sinnvoll erwiesen, entweder  $GF(p)$  mit einer großen Primzahl  $p$  oder  $GF(2^n)$  mit einer (großen) natürlichen Zahl  $n$  zu betrachten. Der Grund für die Verwendung des Primkörpers  $GF(p)$  liegt in seiner einfachen Arithmetik; andererseits kommt  $GF(2^n)$  der binären Darstellung in Computersystemen entgegen. Andere Körper wie z.B.  $GF(7^n)$  bieten keiner dieser beiden Vorteile und werden daher in der Praxis nicht verwendet, ohne dass dies theoretische Gründe hätte.

<sup>11</sup>Karl Weierstrass, 31.10.1815–19.12.1897, deutscher Mathematiker, Verfechter der streng formalen Ausrichtung der Mathematik.

Durch Koordinatentransformation kann man die Weierstrass-Gleichung in einer einfacheren Form schreiben<sup>12</sup>. Je nachdem, ob  $p > 3$  ist, verwendet man unterschiedliche Transformationen und erhält so

- im Fall  $GF(p)$ ,  $p > 3$ , die Elliptische Kurven-Gleichung der Form

$$y^2 = x^3 + ax + b \quad (7.3)$$

mit  $4a^3 + 27b^2 \neq 0$

- im Fall  $GF(2^n)$  die Elliptische Kurven-Gleichung der Form

$$y^2 + xy = x^3 + ax^2 + b \quad (7.4)$$

mit  $b \neq 0$ <sup>13</sup>.

Durch diese Bedingungen an die Parameter  $a, b$  ist gewährleistet, dass die Elliptische Gleichung für kryptographische Anwendungen geeignet ist<sup>14</sup>.

Für die Anzahl  $|E|$  der Elemente einer Elliptischen Kurve  $E$  über einem Körper  $GF(k)$  (praktisch  $k = p$  prim oder  $k = 2^n$ ) gilt nach dem Satz von Hasse [Silverman1986] die einfache Beziehung  $||E| - k| \leq 2 \cdot \sqrt{k}$ . Diese Ungleichung ist äquivalent zu  $k + 1 - 2\sqrt{k} < |E| < k + 1 + 2\sqrt{k}$ . Dies bedeutet, dass die Anzahl der Elemente der Elliptischen Kurve mit der Größe  $k$  gut abgeschätzt werden kann.

## 7.5 Verknüpfung auf Elliptischen Kurven

Um mit Elliptischen Kurven arbeiten zu können, definiert man eine Verknüpfung (meist additiv als  $+$  geschrieben) auf den Punkten der Elliptischen Kurve. Dabei definiert man bei Elliptischen Kurven über  $GF(p)$  die kommutative Verknüpfung durch

1.  $P + \mathcal{O} = \mathcal{O} + P = P$  für alle  $P \in E$ ,
2. für  $P = (x, y)$  und  $Q = (x, -y)$  ist  $P + Q = \mathcal{O}$ ,
3. für  $P_1 = (x_1, y_1), P_2 = (x_2, y_2) \in E$  mit  $P_1, P_2 \neq \mathcal{O}$  und  $(x_2, y_2) \neq (x_1, -y_1)$  ist  $P_3 := P_1 + P_2$ ,  $P_3 = (x_3, y_3)$  definiert durch

$$x_3 := -x_1 - x_2 + \lambda^2, \quad y_3 := -y_1 + \lambda(x_1 - x_3)$$

mit dem Hilfsquotienten

$$\lambda := \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{falls } P_1 \neq P_2, \\ \frac{3x_1^2 + a}{2y_1} & \text{falls } P_1 = P_2. \end{cases}$$

<sup>12</sup>Anschaulich bedeutet eine solche Koordinatentransformation eine Drehung bzw. Streckung der Koordinatenachsen, ohne dass die zugrunde liegende Kurve selbst verändert wird.

<sup>13</sup>Die Form (7.3) ist die Standardform der Weierstrass-Gleichung. Ist die Charakteristik des Körpers jedoch 2 oder 3, so ist  $4 = 0$  bzw.  $27 = 0$ , was dazu führt, dass man in der Bedingung an die Parameter  $a, b$  wesentliche Informationen verliert. Dies ist ein Hinweis darauf, dass die Transformation auf die Standardform in diesen Fällen nicht zu befriedigenden Ergebnissen führt.

<sup>14</sup>Formal sagt man, die Kurve ist nicht singulär.

Hieraus folgt insbesondere für  $P = (x, y) \in E$ , dass gilt  $-P = (x, -y)$ .

Über  $GF(2^n)$  definiert man analog die Verknüpfung durch

1.  $P + \mathcal{O} = \mathcal{O} + P = P$  für alle  $P \in E$ ,
2. für  $P = (x, y)$  und  $Q = (x, x + y)$  ist  $P + Q = \mathcal{O}$ ,
3. für  $P_1 = (x_1, x_2), P_2 = (x_2, y_2) \in E$  mit  $P_1, P_2 \neq \mathcal{O}$  und  $(x_2, y_2) \neq (x_1, x_1 + y_1)$  ist  $P_3 := P_1 + P_2, P_3 = (x_3, y_3)$  definiert durch

$$x_3 := -x_1 + x_2 + \lambda + \lambda^2 + a, \quad y_3 := y_1 + x_3 + \lambda(x_1 + x_3)$$

mit

$$\lambda := \begin{cases} \frac{y_1 + y_2}{x_1 + x_2} & \text{falls } P_1 \neq P_2, \\ x_1 + \frac{y_1}{x_1} & \text{falls } P_1 = P_2. \end{cases}$$

Hieraus folgt insbesondere für  $P = (x, y) \in E$ , dass gilt  $-P = (x, x + y)$ . (Beachte:  $-(-P) = (x, x + (x + y)) = (x, 2x + y) = (x, y)$ , da der zugrunde liegende Körper Charakteristik 2 hat.)<sup>15</sup>

Man kann nachrechnen, dass die Menge  $E \cap \{\mathcal{O}\}$  mit der so definierten Addition eine Gruppe bildet. Dies bedeutet insbesondere, dass die Summe zweier Kurvenpunkte stets wieder ein Punkt auf der Elliptische Kurve ist. Diese Addition lässt sich auch geometrisch veranschaulichen, wie der folgende Abschnitt zeigt.

---

<sup>15</sup>Eine Animation der Punktaddition auf Elliptischen Kurven findet man auf der Certicom-Seite unter [http://www.certicom.com/resources/ecc\\_tutorial/ecc\\_tutorial.html](http://www.certicom.com/resources/ecc_tutorial/ecc_tutorial.html)

## Addieren von Punkten auf einer Elliptischen Kurve

Die zwei folgenden Abbildungen zeigen, wie bei einer Elliptischen Kurve über den reellen Zahlen in affinen Koordinaten zwei Punkte addiert werden. Der unendlich ferne Punkt  $\mathcal{O}$  kann nicht in der affinen Ebene dargestellt werden.

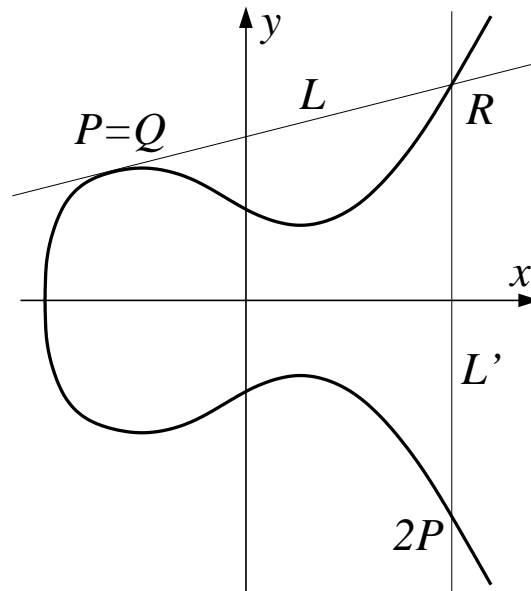


Abbildung 7.4: Verdoppelung eines Punktes

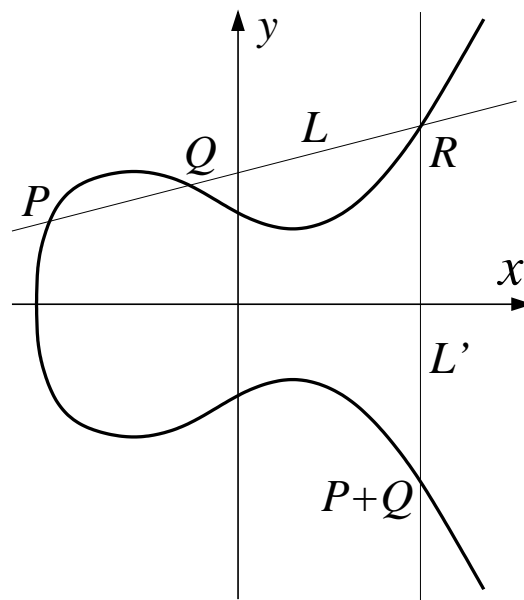


Abbildung 7.5: Addition zweier verschiedener Punkte im Körper der reellen Zahlen



## 7.6 Sicherheit der Elliptischen-Kurven-Kryptographie: Das ECDLP

Wie bereits in Abschnitt 7.4 erwähnt, betrachten wir in der Kryptographie Elliptische Kurven über diskreten<sup>16</sup> Körpern  $GF(2^n)$  oder  $GF(p)$  (für große Primzahlen  $p$ ). Dies bedeutet, dass alle Parameter, die zur Beschreibung der Elliptischen Kurve notwendig sind, aus diesem zugrunde liegenden Körper stammen. Ist nun  $E$  eine Elliptische Kurve über einem solchen Körper und  $P$  ein Punkt auf der Kurve  $E$ , so kann man für jede natürliche Zahl  $m$

$$mP := \underbrace{P + P + \dots + P}_{m \text{ mal}}$$

bilden. Diese Operation ist aus kryptographischer Sicht deshalb besonders interessant, weil man einerseits um  $mP$  zu berechnen im allgemeinen nur  $\log m$  Additionen durchführen muss — man bildet einfach  $P, 2P, 2^2P, 2^3P, \dots$ , schreibt  $m$  binär und addiert schließlich entsprechend der Binärdarstellung von  $m$  auf — es andererseits sehr aufwendig zu sein scheint, zu gegebenen Punkten  $P$  und  $Q = mP$  auf  $E$  die Zahl  $m$  zu bestimmen. Natürlich kann man die Folge  $P, 2P, 3P, 4P, 5P, \dots$  bilden und jeweils mit  $Q$  vergleichen. Hierzu benötigt man jedoch  $m$  Additionen.

Bisher ist noch kein Algorithmus bekannt, der effizient  $m$  aus  $P$  und  $Q$  berechnet. Die bisher besten Verfahren liegen z.B. im Fall  $GF(p)$  in der Größenordnung  $\sqrt{q}$ , wobei  $q$  ein (großer) Primfaktor von  $p - 1$  ist;  $m$  selbst sollte in diesem Fall zwischen 1 und  $q$  liegen, so dass man für die Multiplikation  $mP$  maximal  $\log q$  Schritte benötigt. Der Quotient  $\frac{\sqrt{q}}{\log q}$  strebt jedoch (schnell) gegen  $+\infty$ .

Sind die Parameter hinreichend groß (ist zum Beispiel  $p$  prim und mehr als 160 Bit lang) ist der Computer ohne weiteres in der Lage, sehr schnell (in wenigen Bruchteilen einer Sekunden) den Punkt  $mP$  zu bestimmen. Das *inverse Problem*,  $m$  aus  $mP$  und  $P$  zu erhalten, ist jedoch nicht in akzeptabler Zeit möglich.

Dies wird als das „Diskrete Logarithmus Problem über Elliptischen Kurven“ bezeichnet (auch ECDLP - Elliptic Curve Discrete Logarithm Problem - abgekürzt).

Formal betrachten wir in der Elliptischen-Kurven-Kryptographie die Punkte der Kurve als Elemente einer Gruppe mit der Addition als Verknüpfung. Allerdings sind nur solche Elliptischen Kurven für kryptographische Anwendungen geeignet, bei der die Anzahl der Kurvenpunkte hinreichend groß ist. Ferner können in Spezialfällen Elliptische Kurven auch aus anderen Gründen ungeeignet sein. Das bedeutet, dass man bei der Definition einer Kurve auf die Wahl der Parameter achten muss. Denn für bestimmte Klassen von Elliptischen Kurven ist es möglich, das ECDLP leichter zu lösen als im allgemeinen Fall. Kryptographisch ungeeignete Elliptische Kurven sind die sogenannten *anormalen* Kurven (das sind Kurven über  $\mathbb{Z}_p$ , für die die Menge  $\mathbf{E}$  genau  $p$  Elemente hat) und die *supersingulären* Kurven (das sind Kurven, für die man das Berechnen des ECDLP auf das Berechnen des „normalen“ Diskreten Logarithmus in anderen endlichen Körper reduzieren, d.h. vereinfachen, kann). Daher gibt es kryptographisch gute und schlechte Kurven. Allerdings kann man für gegebene Parameter  $a$  und  $b$  mit etwas Aufwand feststellen, ob die resultierende Elliptische Kurve kryptographisch brauchbar ist oder nicht. Die in der Kryptographie eingesetzten Kurven werden meist von Fachleuten zur Verfügung gestellt. Sie gewährleisten, dass die von ihnen als sicher eingestuften Elliptischen Kurven den aktuellen Sicherheitsanforderungen genügen.

<sup>16</sup>Diskret im Gegensatz zu kontinuierlich.

Bei sicheren Kurven wird hauptsächlich durch den Parameter  $p$  im Fall des zugrunde liegenden Körpers  $GF(p)$  bzw.  $n$  im Fall des zugrunde liegenden Körpers  $GF(2^n)$  bestimmt, wie lange es dauert, das ECDLP auf dieser Kurve zu lösen. Je größer diese Parameter sind, desto länger nimmt das Lösen des Problems in Anspruch. Von Fachleuten wird z.B. eine Bitlänge von über 200 Bit für den Parameter  $p$  empfohlen. Hier wird deutlich, warum die Elliptischen Kurven so interessant für die Kryptographie sind. Denn die Parameter bestimmen auch den Signatur-/Verschlüsselungsaufwand, wenn mit Elliptischen Kurven Kryptographie betrieben wird. Die Dauer einer Schlüsselpaar-Erzeugung ist ebenfalls von den Parametern abhängig. Daher sind kleine Werte (wenige Bits) wünschenswert (möglichst schnelle Laufzeiten der Verfahren); allerdings muss die geforderte Sicherheit dabei eingehalten werden. Mit einer Länge von zum Beispiel 200 Bit für  $p$  ist eine *gute* Elliptische Kurve genau so sicher wie ein RSA-Modulus von über 1024 Bit Länge (zumindest nach dem heutigen Forschungsstand). Der Grund dafür ist, dass die schnellsten Algorithmen zum Lösen des *Elliptische Kurven Diskreter Logarithmus* Problems eine exponentielle Laufzeit haben — im Gegensatz zu den subexponentiellen Laufzeiten, die die zur Zeit besten Faktorisierungsalgorithmen haben (Zahlkörpersieb, Quadratisches Sieb oder Faktorisieren mit Elliptischen Kurven). Dies erklärt, warum die Parameter von Kryptoverfahren, die auf dem Problem *Faktorisieren von ganzen Zahlen* beruhen, größer sind als die Parameter von Kryptoverfahren, die auf dem ECDL-Problem basieren.

## 7.7 Verschlüsseln und Signieren mit Hilfe Elliptischer Kurven

Das *Elliptische Kurven Diskreter Logarithmus Problem* (ECDLP) ist die Grundlage für die Elliptische-Kurven-Kryptographie. Darauf basierend gibt es verschiedene Signaturverfahren. Um ein solches Signaturverfahren anzuwenden, benötigt man:

- Eine Elliptische Kurve  $\mathbf{E}$ , beschrieben durch den zugrunde liegenden Körper  $GF(p^n)$ .
- Eine Primzahl  $q \neq p$  sowie einen Punkt  $G$  auf der Elliptischen Kurve  $\mathbf{E}$  mit Ordnung  $q$ . D.h., es gilt  $qG = \mathcal{O}$  und  $rG \neq \mathcal{O}$  für alle  $r \in \{1, 2, \dots, q-1\}$ . Die Zahl  $q$  muss dann ein Teiler der Gruppenordnung (entspricht der Anzahl der Elemente)  $\#\mathbf{E}$  sein. Aufgrund der Primordnung, erzeugt  $G$  eine zyklischen Untergruppe von  $\mathbf{E}$  mit Ordnung  $q$ .

Die genannten Parameter bezeichnet man als *Domain*-Parameter. Durch sie wird festgelegt, auf welcher Elliptischen Kurve  $\mathbf{E}$  und in welcher zyklischen Untergruppe von  $\mathbf{E}$  ein Signaturverfahren eingesetzt wurde.

### 7.7.1 Verschlüsselung

Mit Hilfe Elliptischer Kurven kann ein sicherer Schlüsselaustausch nach dem Diffie-Hellman Protokoll erfolgen (siehe Kapitel 5.4.2). Dieser Schlüssel kann dann für eine anschließende symmetrische Verschlüsselung verwendet werden. Ein Schlüsselpaar mit privatem und öffentlichem Schlüssel wird im Gegensatz zum RSA-Algorithmus hingegen nicht erzeugt!

In der Schreibweise der Elliptischen Kurven liest sich das Diffie-Hellman Verfahren wie folgt: Zunächst einigen sich beide Partner (A und B) öffentlich auf eine Gruppe  $G$  und eine ganze Zahl  $q$ . Danach wählen sie zufällig  $r_A, r_B \in \{1, 2, \dots, q-1\}$ , bilden die Punkte  $R_A = r_A G$ ,  $R_B = r_B G$  auf der Elliptischen Kurve und tauschen diese aus. Danach berechnet A leicht  $R = r_A R_B$ . Denselben Punkt (nämlich  $r_A r_B G$ ) erhält auch B, indem er  $r_B R_A = r_B r_A G = r_A r_B G = R$  bildet. Dabei ist die Berechnung von  $R_A, R_B$  als  $r_A$  bzw.  $r_B$ -faches des Kurvenpunktes  $G$  leicht

durchzuführen; die umgekehrte Operation, aus  $R_A$  bzw.  $R_B$  den Wert  $r_A$  bzw.  $r_B$  zu erhalten, ist jedoch sehr aufwändig.

Für einen Dritten ist es nach heutigem Kenntnisstand nicht möglich,  $R$  zu berechnen, wenn er nicht mindestens einen der Werte  $r_A$  oder  $r_B$  ermitteln kann, d.h. das ECDLP löst.

Um einen “Man-in-the-Middle”-Angriff zu verhindern, kann man auch hier wie schon in Kapitel 6.4.1 beschrieben, die übertragenen Werte  $G, q, R_A, R_B$  digital signieren.

### 7.7.2 Signatur-Erstellung

Überträgt man den DSA auf Elliptische Kurve, so kann man wie folgt eine digitale Signatur erzeugen: Man wählt vorab eine (nicht-triviale) Zahl  $s \in \mathbb{Z}_q$ . Diese bildet den privaten Schlüssel. Hingegen werden  $q, G$  und  $R = sG$  veröffentlicht. Aus  $G$  und  $R$  lässt sich jedoch  $s$  nicht ermitteln, worauf die Sicherheit des Signaturverfahrens beruht.

Für eine Nachricht  $m$  wird zunächst mit Hilfe eines Hash-Verfahrens  $h$  ein digitaler Fingerabdruck erstellt, wobei  $h(m)$  im Wertebereich  $\{0, 1, 2, \dots, q-1\}$  liegt und  $h(m)$  somit als Element von  $\mathbb{Z}_q$  interpretiert werden kann. Dann wird ein zufälliges  $r \in \mathbb{Z}_q$  gewählt und  $R = (r_1, r_2) = rG$  berechnet. Die erste Komponente  $r_1$  von  $R$  ist ein Element von  $GF(p^n)$ . Diese wird auf  $\mathbb{Z}_q$  abgebildet, z.B. im Fall  $n = 1$  als Element von  $\{0, 1, \dots, p-1\}$  interpretiert und dann der Teilerrest modulo  $q$  gebildet. Das so erhaltene Element von  $\mathbb{Z}_q$  bezeichnen wir mit  $\bar{r}_1$ . Nun bestimmt man  $x \in \mathbb{Z}_q$  mit

$$rx - s\bar{r}_1 - h(m) = 0.$$

Das Tripel  $(m, r_1, x)$  bildet nun die digitale Signatur.

### 7.7.3 Signatur-Verifikation

Zur Verifikation muss zunächst  $u_1 = h(m)/x, u_2 = \bar{r}_1/x$  (in  $\mathbb{Z}_q$  gebildet werden). Dann bestimmt man

$$V = u_1G + u_2Q.$$

Wegen  $Q = sG$  ist  $V = (v_1, v_2)$  mit  $v_1 = u_1 + u_2s$ . Diese Addition findet formal im Raum  $GF(p^n)$  statt. Die Projektion von  $GF(p^n)$  auf  $\mathbb{Z}_q$  sollte jedoch so gewählt sein, dass  $\bar{v}_1 = u_1 + u_2s$  in  $\mathbb{Z}_q$  ist. Dann gilt nämlich

$$\bar{v}_1 = u_1 + u_2s = h(m)/x + \bar{r}_1s/x = (h(m) + \bar{r}_1s)/x = rx/x = r.$$

Nun ist  $R = rG$ . Also folgt hier  $\bar{v}_1 = \bar{r}_1$ , d.h.  $R$  und  $V$  stimmen modulo der Projektion auf  $\mathbb{Z}_q$  überein.

## 7.8 Faktorisieren mit Elliptischen Kurven

Es gibt Faktorisierungsalgorithmen<sup>17</sup>, die auf Elliptischen Kurven basieren<sup>18</sup>. Genauer gesagt, machen sich diese Verfahren zunutze, dass man auch über  $\mathbb{Z}_n$  ( $n$  zusammengesetzte Zahl) El-

<sup>17</sup> Insbesondere John M. Pollard war an der Entwicklung vieler verschiedener Faktorisierungsalgorithmen beteiligt; auch beim Faktorisieren mit ECC war er einer der führenden Köpfe. Als Mitarbeiter von British Telecom hat er leider nie viel selbst publiziert. Auf der RSA Konferenz 1999 wurde er für seine „outstanding contributions in mathematics“ ausgezeichnet:

[http://www.ietf.org/Privacy/Crypto\\_misc/DESCracker/HTML/19990118\\_rsa\\_awards.html](http://www.ietf.org/Privacy/Crypto_misc/DESCracker/HTML/19990118_rsa_awards.html).

<sup>18</sup> Im Jahr 1987 stellte H.W. Lenstra einen Faktorisierungsalgorithmus vor, der auf Elliptischen Kurven basiert (siehe [Lenstra1987]). Die aktuell größte mit Elliptischen Kurven faktorisierte Zahl ist die 55 Dezimalstellen lange Zahl  $629^{59} - 1$ , sie wurde am 6. Oktober 2001 von M. Izumi gefunden (siehe ECMNET).

liptische Kurven definieren kann. Elliptische Kurven über  $\mathbb{Z}_n$  bilden keine Gruppe, da es nicht zu jedem Punkt auf solchen Elliptischen Kurven einen inversen Punkt geben muss. Dies hängt damit zusammen, dass es – falls  $n$  eine zusammengesetzte Zahl ist – in  $\mathbb{Z}_n$  Elemente gibt, die kein Inverses bezüglich der Multiplikation modulo  $n$  haben. Um zwei Punkte auf einer Elliptischen Kurve über  $\mathbb{Z}_n$  zu addieren, kann prinzipiell genauso gerechnet werden wie auf Elliptischen Kurven über  $\mathbb{Z}_p$ . Eine Addition von zwei Punkten (auf einer Elliptischen Kurve über  $\mathbb{Z}_n$ ) scheitert aber genau dann, wenn man einen Teiler von  $n$  gefunden hat. Der Grund dafür ist, dass das Verfahren zum Addieren von Punkten auf Elliptischen Kurven Elemente in  $\mathbb{Z}_n$  ermittelt und zu diesen Elementen die inversen Elemente (bezüglich der Multiplikation modulo  $n$ ) in  $\mathbb{Z}_n$  berechnet. Dazu wird der erweiterte Euklidische Algorithmus benutzt. Ergibt sich nun bei der Addition zweier Punkte (die auf einer Elliptischen Kurve über  $\mathbb{Z}_n$  liegen) ein Element aus  $\mathbb{Z}_n$ , das kein inverses Element in  $\mathbb{Z}_n$  hat, so gibt der erweiterte Euklidische Algorithmus einen echten Teiler von  $n$  aus.

Das Faktorisieren mit Elliptischen Kurven funktioniert somit prinzipiell so: Man wählt zufällige Kurven über  $\mathbb{Z}_n$ , sowie zufällig irgendwelche Punkte (die auf diesen Kurve liegen) und addiert diese; dabei bekommt man wieder Punkte, die auf der Kurve liegen oder findet einen Teiler von  $n$ . Die Faktorisierungsalgorithmen auf Basis von Elliptischen Kurven arbeiten also probabilistisch. Durch die Möglichkeit, sehr viele Elliptische Kurven über  $\mathbb{Z}_n$  zu definieren, kann man die Wahrscheinlichkeit erhöhen, zwei Punkte zu finden, bei deren Addition ein Teiler von  $n$  gefunden wird. Daher eignen sich diese Verfahren auch sehr gut für eine Parallelisierung.

## 7.9 Implementierung Elliptischer Kurven zu Lehrzwecken

Es gibt relativ wenig freie Programme mit graphischer Oberfläche, die ECC implementieren. Im Folgenden wird aufgezeigt, welche Funktionalität dazu in CrypTool und in Sage vorhanden ist.

### 7.9.1 CrypTool

CrypTool enthält Elliptische Kurven, um digitale Signaturen zu erzeugen<sup>19</sup> und um die ECC-AES-Hybridverschlüsselung durchzuführen<sup>20</sup>.

Implementiert sind die Basisalgorithmen für Gruppenoperationen, für das Erzeugen von Elliptischen Kurven und für das Ein- und Auslesen von Parametern für Elliptische Kurven über endlichen Körpern  $GF(p)$  mit  $p$  Elementen ( $p$  prim). Die Implementierung erfolgte in ANSI C und richtete sich nach dem Entwurf Nr. 8 der Arbeitsgruppe IEEE P1363 *Standard Specifications for Public Key Cryptography*

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

Implementiert sind die kryptographischen Primitive zur Signaturerzeugung und Signaturverifikation für die auf Elliptischen Kurven basierenden Varianten von Nyberg-Rueppel-Signaturen und DSA-Signaturen.

### 7.9.2 Sage

In Sage werden Elliptische Kurven beschrieben unter

[http://www.sagemath.org/doc/constructions/elliptic\\_curves.html](http://www.sagemath.org/doc/constructions/elliptic_curves.html)<sup>21</sup>.

Zusätzlich gibt es ein ausführliches, interaktives ECC-Tutorial von Maike Massierer. Diese Einführung in die Elliptische-Kurven-Kryptographie (ECC) ist als Sage-Notebook aufgebaut.

Sage-Notebooks werden im Browser nach einem Logon-Vorgang aufgerufen<sup>22,23</sup>.

---

<sup>19</sup>Die Dialogbox, die in CrypTool nach dem Menü **Digitale Signaturen/PKI \ Dokument signieren** erscheint, bietet die EC-Verfahren ECSP-DSA und ECSP-NR an.

<sup>20</sup>In CrypTool finden Sie dieses Verfahren über das Menü **Ver-/Entschlüsseln \ Hybrid**.

<sup>21</sup>Sage-Beispiele dazu finden sich z.B. unter den „Published Worksheets“ auf <http://www.sagenb.org/pub/>:

- über Elliptic Curve: <http://www.sagenb.org/home/pub/606/>
- über Elliptic Curve El Gamal: <http://www.sagenb.org/home/pub/104/>, oder im
- das „Elliptic Curve Cryptography (ECC) Tutorial“  
<http://www.williamstein.org/simuw06/notes/notes/node12.html>

<sup>22</sup>Hat man Sage auf einem eigenen (Unix-)Server installiert, muss man auf der Sage-Kommandozeile erst den Befehl `notebook()` aufrufen.

<sup>23</sup>Das ECC-Notebook von Maike Massierer benötigt die KASH3-Bibliothek: Deshalb muss (z.B. für Sage 4.2.1) das Package „kash3-2008-07-31.spkg“ installiert worden sein (Befehl `sage -i`).

Das ECC-Notebook von Massierer<sup>24,25</sup> besteht aus 8 Teilen („Titelseite“ mit Inhaltsverzeichnis plus 7 Kapitel) und zielt darauf ab, dass selbst ein Einsteiger versteht, was Elliptische Kurven sind (es ist nur auf Englisch vorhanden):

0. ECC Notebook (title page and contents)
1. Introduction and Overview
2. Motivation for the use of Elliptic Curves in Cryptography
3. Elliptic Curves in Cryptography
4. Cryptographic Protocols in ECC
5. Domain Parameter Generation for ECC Systems
6. Conclusion and Further Topics
7. References

---

<sup>24</sup> Anleitung zur Benutzung eines interaktiven Sage-Notebooks:

- Öffentliche Sage-Server wie <http://sage.mathematik.uni-siegen.de:8000> oder <http://www.sagenb.org/> bieten oft Worksheets als „Published Worksheets“ an, die man ohne Log-in ausführen kann. Diese Worksheets werden aufgelistet, wenn man auf „Published“ in der oberen rechten Ecke klickt.
- Worksheets, die den **Interact**-Befehl nutzen, erfordern z.Zt. einige weitere Schritte vom Benutzer: Einloggen, Kopie erstellen, alle Kommandos nochmal ausführen.  
Das Vorgehen ist im Folgenden beschrieben (am Beispiel des sagenb-Servers und für das ECC-Tutorial):
- Anlegen eines Accounts für ein Sage-Notebook unter <http://sagenb.org/register> und Einloggen unter <http://sagenb.org/>.
- Öffnen des Worksheets <http://sagenb.org/home/pub/1126/>. Dieses enthält das Inhaltsverzeichnis des interaktiven ECC-Notebook. Von hier aus kann man per Klick zu den anderen Kapiteln des Dokuments navigieren.
- Klicken Sie auf **Edit a copy** in der linken oberen Ecke, um eine eigene Kopie des Worksheets zu erstellen.
- Manchmal ist es nötig, dass man das Worksheet nach dem Start nochmal neu ausführt. Klicken Sie dazu in der linken oberen Ecke **Action -> Evaluate all**.
- Manche Befehle funktionieren manchmal trotzdem nicht nach dem Öffnen eines Worksheet. Statt eines ansprechenden Layouts kommen viele (blaue) Fehlermeldungen. Das kann man normalerweise schnell lösen, indem man den grauen „%hide“-String anklickt: Danach sieht man den Code hinter der Grafik. Mit Shift-Enter kann man die Grafik neu erzeugen.  
Selbst dann verschwindet der Grafik-Code nicht immer, sondern wird grau. Dann hilft meist ein Klick auf den grauen Text, und danach ein Klick außerhalb der Text-Box. Danach verschwindet der Code und man sieht das grafische Layout des Worksheet.
- Teile des ECC-Tutorial benutzen einen speziellen Mathematik-Font, der standardmäßig bei den meisten Browser nicht mitinstalliert wird. Wenn Sie bemerken, dass Formeln nicht korrekt dargestellt sind oder Ihr Browser meldet, dass Fonts fehlen, installieren Sie bitte die Fonts jsMath für eine bessere Darstellung.  
Siehe <http://www.math.union.edu/~dpvc/jsMath/> und <http://pubpages.unh.edu/~jsh3/jsMath/>.  
Nach der Installation dieser Fonts hat man das jsMath-Symbol am unteren Rand des Browsers. Klickt man dieses Symbol an, erhält man die Download-Seite dieser TIFF-Fonts. Diese Font-Installation muss an jedem PC einzeln erfolgen.
- Gemäß der Sage-Support-Newsgroup wird daran gearbeitet, ein System zu erstellen, dass **@interact** komplett außerhalb des Sage-Notebooks benutzt werden kann (JS-Code innerhalb von statischen HTML-Seiten).

<sup>25</sup> Seit 2008 steht dieses ECC-Notebook auf <http://sage.mathematik.uni-siegen.de:8000/home/pub/45/> (#45 bis #52). Um sich in Siegen einzuloggen, muss man den Port 8000 und Cookies erlauben.  
Seit 2009 gibt es eine aktualisierte Version dieses ECC-Notebook auf <http://sagenb.org/home/pub/1126/> (#1126 bis #1133).

## 7.10 Patentaspekte

Wenn man statt des Primkörpers  $GF(p)$  einen Körper der Form  $GF(2^n)$  zugrunde legt, ergeben sich wesentliche Unterschiede in der Implementierung. Der Vorteil einer Implementierung unter Verwendung von  $GF(2^n)$  liegt darin, dass Rechnungen aufgrund der binären Darstellung effizienter durchgeführt werden können. Dies betrifft insbesondere die Division, die in  $GF(p)$  vergleichsweise aufwändig ist (was z.B. bei dem oben beschriebenen Signaturverfahren sowohl die Erstellung der Signatur als auch ihre spätere Verifikation betrifft, da beide eine Divisionsoperation enthalten).

Um das Potenzial der Effizienzsteigerung möglichst optimal zu nutzen, kann man z.B. Körper wählen, die besondere Basen besitzen, wie Polynomialbasen (besonders geeignet für Software-Implementierungen) oder Normalbasen (bevorzugt bei Hardware-Implementierungen). Für bestimmte Werte von  $n$  (wie z.B.  $n = 163, 179, 181$ ) lassen sich sogar beide Vorteile kombinieren. Allerdings sind spezielle Darstellungen oft nicht Standard.

Um Speicherplatz zu sparen, wird zuweilen nur die erste Komponente sowie ein weiteres Bit für jeden Punkt auf der Elliptischen Kurve gespeichert. Hieraus kann jeweils der gesamte Punkt errechnet werden. Dies ist besonders bei Verwendung von Normalbasen effizient. Selbst bei der Durchführung der kryptographischen Protokolle kann eine signifikante Beschleunigung erreicht werden. Diese sogenannte *Punkt-Kompression*, die bei der Hälfte aller Kurven einsetzbar ist, ist jedoch patentiert (US Patent 6141420, Certicon) und daher nicht ohne weiteres frei einsetzbar.

Im allgemeinen Fall  $GF(p^n)$  (aber auch für  $n = 1$ ) werden oft sogenannte affine oder projektive Koordinaten eingesetzt, was je nach Anwendung zu Effizienzgewinnen führt.

Eine vollständige Beschreibung aller Implementierungen unter Abwägung ihrer Vor- und Nachteile würde an dieser Stelle zu weit führen. Abschließend kann festgehalten werden, dass die Vielfalt möglicher Implementierungen bei Elliptischen Kurven z.B. im Vergleich zu RSA-Implementierungen sehr groß ist. Aus diesem Grund gibt es Bestrebungen, sich auf wenige Standardimplementierungen, ja sogar auf eine kleine Schar fest vorgegebener Kurven zu beschränken (ASC-Ansatz).

Der Erfolg dieser Standardisierungsbemühungen ist heute noch nicht absehbar. Dies wäre aber eine Voraussetzung dafür, dass sich ECC dauerhaft als Alternative zu RSA etabliert. Die Arbeit der Standardisierungskomitees wird sich erheblich beschleunigen müssen, wenn sich abzeichnet, dass die aktuellen Forschungsprojekte im Bereich der Faktorisierung einen Durchbruch erzielen.

## 7.11 Elliptische Kurven im praktischen Einsatz

Bereits heute werden Elliptische Kurven in der Praxis eingesetzt. Als prominentes Beispiel ist hier der Informationsverbund Bonn-Berlin (IVBB)<sup>26</sup> zu nennen, bei dem streng vertrauliche Dokumente der deutschen Bundesregierung zwischen Regierungsstellen mit Sitz in Berlin und Bonn ausgetauscht werden. Durch den Einsatz von ECC konnte eine Hochsicherheitslösung realisiert werden. Fragen der Interoperabilität spielten hingegen eine untergeordnete Rolle.

In Österreich gibt es eine Massenanwendung auf Basis von ECC: Die Bankkarte mit Signaturfunktion.

---

<sup>26</sup>Der IVBB verbindet Regierungsstellen in der alten und neuen deutschen Hauptstadt.  
[http://www.cio.bund.de/cIn\\_094/sid\\_92C19118CBA5A021AFD1ABAEC15D2B77/DE/IT-Angebot/IT-Infrastrukturen/IVBB/ivbb\\_inhalt.html](http://www.cio.bund.de/cIn_094/sid_92C19118CBA5A021AFD1ABAEC15D2B77/DE/IT-Angebot/IT-Infrastrukturen/IVBB/ivbb_inhalt.html)

Beide Beispiele zeigen gut die typischen Einsatzgebiete Elliptischer Kurven: Als Hochsicherheitslösungen und bei Implementierungen auf Smartcards, bei denen der Schlüssellänge (Speicherplatz) eine entscheidende Bedeutung zukommt.



# Literaturverzeichnis

- [Cassels1991] J. W. S. Cassels,  
*Lectures on elliptic curves*,  
Cambridge University Press, 1991, 143 Seiten.
- [Koblitz1984] N. Koblitz,  
*Introduction to elliptic curves and modular forms*,  
Graduate Texts in Mathematics, Springer-Verlag, 1984.
- [Koblitz1998] N. Koblitz,  
*Algebraic aspects of Cryptography. With an appendix on Hyperelliptic curves by Alfred J. Menezes, Yi Hong Wu and Robert J. Zuccherato*,  
Springer-Verlag, 1998, 206 Seiten.
- [Lenstra1987] H.W. Lenstra,  
*Factoring integers with elliptic curves*,  
Annals of Mathematics 126, pp. 649-673, 1987.
- [Lenstra1999] Arjen K. Lenstra, Eric R. Verheul,  
*Selecting Cryptographic Key Sizes (1999)*,  
Journal of Cryptology: the journal of the International Association for Cryptologic Research  
<http://www.cryptosavvy.com/cryptosizes.pdf>
- [Menezes1993] A. J. Menezes,  
*Elliptic curve public key cryptosystems*,  
Kluwer Academic Publishers, 1993.
- [Silverman1986] J. Silverman,  
*The Arithmetic of Elliptic Curves*,  
Springer-Verlag, 1986.
- [Silverman1992] J. Silverman,  
*The arithmetic of elliptic curves*,  
Graduate Texts in Mathematics, Springer-Verlag, 1992.
- [SilvermanTate1992] J. Silverman, J. Tate,  
*Rational points on elliptic curves*,  
Springer-Verlag, 1992.

# Web-Links

1. Umfassende interaktive Einführung in elliptische Kurven und Elliptische-Kurven-Kryptographie (ECC) mit Sage von Maike Massierer und dem CrypTool-Team (alles in Englisch),  
<http://sagenb.org/home/pub/1126/> (#1126 bis #1133)  
ECC-Tutorial als Sage Notebook  
Version 1.2, November 2009
2. Online-Tutorial über Elliptische Kurven der Firma Certicom,  
[http://www.certicom.com/resources/ecc\\_tutorial/ecc\\_tutorial.html](http://www.certicom.com/resources/ecc_tutorial/ecc_tutorial.html)
3. Arbeitsgruppe IEEE P1363,  
<http://grouper.ieee.org/groups/1363>
4. Eine informative Seite zum Faktorisieren mit Elliptischen Kurven,  
<http://www.loria.fr/~zimmerma/records/ecmnet.html>  
Dort findet man Literatur zum Thema Faktorisieren mit Elliptischen Kurven sowie Links zu anderen ECC-Seiten.
5. Schlüssellängen-Vergleich von Arjen Lenstra und Eric Verheul,  
<http://cryptosavvy.com/table.htm>

## Kapitel 8

# Krypto 2020 — Perspektiven für langfristige kryptographische Sicherheit

(Johannes Buchmann, Erik Dahmen, Alexander May und Ulrich Vollmer, TU Darmstadt, Mai 2007)

Kryptographie ist ein Grundbaustein aller IT-Sicherheitslösungen. Aber wie lange sind die heute benutzten kryptographischen Verfahren noch sicher? Reicht diese Zeit, um zum Beispiel medizinische Daten lang genug geheim zu halten? Doch auch kurzfristig ließe sich großer Schaden anrichten, wenn auch nur bestimmte Schlüssel gebrochen würden: Etwa bei den digitalen Signaturen, welche die Authentizität von automatischen Windows-Updates sichern.

### 8.1 Verbreitete Verfahren

In ihrer berühmten Arbeit aus dem Jahr 1978 stellten Rivest, Shamir und Adleman [7] das RSA Public-Key-Verschlüsselungs- und Signaturverfahren vor. RSA ist auch heute noch das in der Praxis meistverwendete Public-Key-System. Die Sicherheit von RSA beruht auf der Schwierigkeit, so genannte RSA-Moduln  $N = pq$  in ihre (großen) Primfaktoren  $p$  und  $q$  zu zerlegen. In ihrer Arbeit schlugen die Erfinder von RSA damals vor, für langfristige Sicherheit 200-stellige RSA-Moduln zu verwenden. Später veröffentlichte die Firma RSA Security eine Liste von RSA-Moduln wachsender Größe (RSA Challenge Numbers) und setzte Preise von insgesamt 635.000 US-\$ für das Faktorisieren dieser Zahlen aus (siehe [www.rsasecurity.com/rsalabs/](http://www.rsasecurity.com/rsalabs/)).

Im Jahr 2005, also 27 Jahre nach der Erfindung von RSA, gelang es Bahr, Boehm, Franke und Kleinjung von der Universität Bonn bereits innerhalb von fünf Monaten eine 200-stellige RSA-Challenge zu faktorisieren und damit die ursprüngliche Empfehlung langfristiger Sicherheit zu brechen ([www.mat.uniroma2.it/~eal/rsa640.txt](http://www.mat.uniroma2.it/~eal/rsa640.txt)). Dies belegt anschaulich die Fortschritte den letzten 30 Jahre bei der Faktorisierung von RSA-Moduln. Diese beruhen sowohl auf bahnbrechenden mathematischen Ideen, zum Beispiel der Entwicklung des Zahlkörpersiebs durch John Pollard, als auch auf bedeutenden Fortschritten in der Computer- und Implementierungstechnik.<sup>1</sup>

Lenstra und Verheul entwickelten 2000 eine Interpolationsformel zur Voraussage der Sicher-

---

<sup>1</sup>Vergleiche Kapitel 4.11 Zur Sicherheit des RSA-Verfahrens, und speziell die Kapitel 4.11.4 und 4.11.5.

heit von RSA und anderen wichtigen kryptographischen Verfahren (vgl. [www.keylength.com](http://www.keylength.com)). Dieser Formel zufolge muss man derzeit schon 850-stellige RSA-Moduln verwenden, um Sicherheit für die nächsten dreißig Jahre zu gewährleisten.

Aber auch eine solche Interpolationsformel ist keine Sicherheitsgarantie! Brillante mathematische Ideen könnten jederzeit dazu führen, dass das Lösen des Faktorisierungsproblems leicht und RSA damit generell unbrauchbar wird. So bewies beispielsweise Peter Shor 1996, dass ein Quantencomputer – ein neuer Computertyp, der die Gesetze der Quantenmechanik ausnutzt – große Zahlen im Prinzip sehr schnell faktorisieren könnte [8]. Trotz intensiver Forschungsbemühungen ist es aber auch heute noch unklar, ob sich jemals hinreichend leistungsfähige Quantencomputer bauen lassen.<sup>2</sup> Aktuelle Verlautbarungen der Start-up-Firma D-Wave ([www.dwavesys.com](http://www.dwavesys.com)) trafen auf verbreitete Skepsis, ja Spott.

Analog zu RSA verläuft die Entwicklung bei Angriffen auf die meistverwendeten Public-Key-Alternativen: den Digital Signature Algorithm (DSA) und Elliptic Curve Cryptography (ECC), die beide auf der Schwierigkeit der Berechnung diskreter Logarithmen beruhen. Es gibt schon heute deutliche algorithmische Fortschritte und Quantencomputer würden auch diese Verfahren unsicher machen.

Wie steht es um die langfristige Sicherheit von so genannten Secret-Key-Verschlüsselungsverfahren? DES wurde 1977 als Data Encryption Standard eingeführt [9] – 21 Jahre später stellte die Electronic Frontier Foundation (EFF) den Spezialcomputer Deep Crack vor, der DES in 56 Stunden bricht. Das Problem von DES war die zu kurz gewählte Schlüssellänge: Offenbar hatten seine Erfinder die rasante Entwicklung bei der Hardware nicht richtig einkalkuliert. Der DES-Nachfolger Advanced Encryption Standard (AES) [6] gilt heute als sicher, wenngleich interessante Angriffsversuche mit algebraischen Methoden existieren.

## 8.2 Vorsorge für morgen

Ist die heutige Kryptographie angesichts ihrer wachsenden Bedeutung noch sicher genug? Die Erfahrung zeigt: Sorgfältig konzipierte und implementierte kryptographische Verfahren haben eine Lebensdauer von 5 bis 20 Jahren. Wer heute RSA, ECC oder AES zur kurzfristigen Absicherung verwendet, darf sich sicher fühlen. Und langfristige Verbindlichkeit lässt sich beispielsweise mit den von Jan Sönke Maseberg vorgeschlagenen multiplen Signaturen lösen [3].

Aber langfristige Vertraulichkeit können wir heute mit den genannten Verfahren nicht garantieren. Und was ist in 20 Jahren? Was kann man tun, wenn ein unerwarteter mathematischer Fortschritt ein wichtiges Kryptoverfahren plötzlich – quasi über Nacht – unsicher macht? Drei Dinge sind zur Vorbereitung nötig:

- ein Pool alternativer sicherer Kryptoverfahren,

<sup>2</sup>Benötigte qbits für Angriffe auf RSA, DSA und ECDSA für Schlüssel der Bit-Länge  $n$ :

RSA	$2n + 3$
DSA	$2n + 3$
ECDSA $2^n$	$\sim 2n + 8 \log n$
ECDSA $p$	$\sim 4n$

Vergleiche Kap. 5.3 in „SicAri – Eine Sicherheitsplattform und deren Werkzeuge für die ubiquitäre Internetnutzung, KB2.1 – Abschlussbericht, Übersicht über Angriffe auf relevante kryptographische Verfahren“, Version 1.0, 17. Mai 2005, Prof. Dr. Johannes Buchmann et al., TUD-KryptC und cv cryptovision GmbH ([http://www.cdc.informatik.tu-darmstadt.de/~schepers/kb\\_21\\_angriffe.pdf](http://www.cdc.informatik.tu-darmstadt.de/~schepers/kb_21_angriffe.pdf)) und die Dissertation von Axel Schmidt am gleichen Lehrstuhl.

- Infrastrukturen, die es ermöglichen, unsichere kryptographische Verfahren leicht gegen sichere auszutauschen und
- Verfahren, die langfristige Vertraulichkeit sicherstellen.

Diese Ziele verfolgen auch die Kryptoarbeitsgruppe der TU Darmstadt und der daraus entstandene Spin-off FlexSecure ([www.flexsecure.de](http://www.flexsecure.de)) seit vielen Jahren. Die Trustcenter-Software FlexiTrust, die in der deutschen nationalen Root-Zertifizierungsstelle und in der deutschen Country-Signing-Zertifizierungsstelle verwendet wird, ist eine Infrastruktur, die den einfachen Austausch von Kryptographie-Verfahren möglich macht. Die Open-Source-Bibliothek FlexiProvider (siehe [www.flexiprovider.de](http://www.flexiprovider.de)) stellt zudem eine Vielzahl unterschiedlicher kryptographischer Verfahren zur Verfügung und in jüngerer Zeit laufen intensive Forschungen zur “Post Quantum Cryptography”: Kryptographie, die auch dann noch sicher bleibt, wenn es (leistungsfähige) Quantencomputer tatsächlich gibt.

Die Sicherheit der Public-Key-Kryptographie beruht traditionell auf der Schwierigkeit, bestimmte mathematische Probleme zu lösen. Heute diskutierte Alternativen zum Faktorisierungs- und Diskrete-Logarithmen-Problem sind: das Dekodierungsproblem, das Problem, kürzeste und nächste Gittervektoren zu berechnen, und das Problem, quadratische Gleichungen mit vielen Variablen zu lösen. Es wird vermutet, dass diese Probleme auch von Quantencomputern nicht effizient lösbar wären.

### 8.3 Neue mathematische Probleme

Wie sehen diese Alternativen näher aus? Verschlüsselung auf der Basis des Dekodierungsproblems wurde von McEliece erfunden [4]. Der Hintergrund: Fehlerkorrigierende Codes dienen dazu, digitale Informationen so zu speichern, dass sie selbst dann noch vollständig lesbar bleiben, wenn einzelne Bits auf dem Speichermedium verändert werden. Diese Eigenschaft nutzen zum Beispiel CDs, sodass Informationen auf leicht verkratzten Datenträgern immer noch vollständig rekonstruierbar sind.

Bei der codebasierten Verschlüsselung werden Daten verschlüsselt, indem zu ihrer Kodierung mit einem öffentlich bekannten Code gezielt Fehler addiert werden, das heißt einzelne Bits werden verändert. Die Entschlüsselung erfordert die Kenntnis einer geeigneten Dekodierungsmethode, welche diese Fehler effizient zu entfernen vermag. Diese Methode ist der geheime Schlüssel – nur wer sie kennt, kann dechiffrieren. Codebasierte Public-Key-Verschlüsselung ist im Allgemeinen sehr effizient durchführbar. Derzeit wird daran geforscht, welche Codes zu sicheren Verschlüsselungsverfahren mit möglichst kleinen Schlüsseln führen.

Verschlüsselung auf der Grundlage von Gitterproblemen ist den codebasierten Verschlüsselungsverfahren sehr ähnlich. Gitter sind regelmäßige Strukturen von Punkten im Raum. Zum Beispiel bilden die Eckpunkte der Quadrate auf kariertem Papier ein zweidimensionales Gitter. In der Kryptographie verwendet man jedoch Gitter in viel höheren Dimensionen. Verschlüsselt wird nach dem folgenden Prinzip: Aus dem Klartext wird zunächst ein Gitterpunkt konstruiert und anschließend geringfügig verschoben, sodass er kein Gitterpunkt mehr ist, aber in der Nähe eines solchen liegt. Wer ein Geheimnis über das Gitter kennt, kann diesen Gitterpunkt in der Nähe finden und damit entschlüsseln. Ein besonders praktikables Gitterverschlüsselungsverfahren ist NTRU ([www.ntru.com](http://www.ntru.com)). Da NTRU vor vergleichsweise kurzer Zeit eingeführt wurde (1998) und seine Spezifizierung aufgrund verschiedener Angriffe mehrfach geändert wurde, sind allerdings noch weitere kryptanalytische Untersuchungen erforderlich, um Vertrauen in dieses Verfahren zu gewinnen.

## 8.4 Neue Signaturen

1979 schlug Ralph Merkle einen bemerkenswerten Ansatz für die Konstruktion von Signaturverfahren in seiner Dissertation [5] vor. Im Gegensatz zu allen anderen Signaturverfahren beruht seine Sicherheit nicht darauf, dass ein zahlentheoretisches, algebraisches oder geometrisches Problem schwer lösbar ist. Benötigt wird ausschließlich, was andere Signaturverfahren ebenfalls voraussetzen: eine sichere kryptographische Hash-Funktion und ein sicherer Pseudozufallsgenerators. Jede neue Hash-Funktion führt somit zu einem neuen Signaturalgorithmus, wodurch das Merkle-Verfahren das Potential hat, das Problem der langfristigen Verfügbarkeit digitaler Signaturverfahren zu lösen.

Merkle verwendet in seiner Konstruktion so genannte Einmal-Signaturen: Dabei benötigt jede neue Signatur einen neuen Signierschlüssel und einen neuen Verifikationsschlüssel. Die Idee von Merkle ist es, die Gültigkeit vieler Verifikationsschlüssel mittels eines Hash-Baums auf die Gültigkeit eines einzelnen öffentlichen Schlüssels zurückzuführen. Im Merkle-Verfahren muss man bei der Schlüsselerzeugung eine Maximalzahl möglicher Signaturen festlegen, was lange wie ein gravierender Nachteil aussah. In [2] wurde jedoch eine Variante des Merkle-Verfahrens vorgestellt, die es ermöglicht, äußerst effizient mit einem einzigen Schlüsselpaar bis zu  $2^{40}$  Signaturen zu erzeugen und zu verifizieren.

## 8.5 Quantenkryptographie – Ein Ausweg?

Ungelöst bleibt aus Sicht der heutigen Kryptographie das Problem der langfristigen Vertraulichkeit: Ein praktikables Verfahren, das die Vertraulichkeit einer verschlüsselten Nachricht über einen sehr langen Zeitraum sicherstellt, ist derzeit nicht bekannt.

Einen Ausweg kann hier möglicherweise die Quantenkryptographie liefern: Sie ermöglicht Verfahren zum Schlüsselaustausch (sehr lange Schlüssel für One-time-Pads), deren Sicherheit auf der Gültigkeit der Gesetze der Quantenmechanik beruhen, vgl. z.B. [1]. Jedoch sind bekannte Verfahren der Quantenkryptographie derzeit noch sehr ineffizient und es ist unklar, welche kryptographischen Funktionen damit realisiert werden können.

## 8.6 Fazit

Wie lautet die Bilanz? Die heutige Kryptographie liefert gute Werkzeuge, um kurzfristige und mittelfristige Sicherheit zu gewährleisten. Anwendungen können diese Werkzeuge ruhigen Gewissens verwenden, solange sie in der Lage sind, unsichere Komponenten schnell gegen Alternativen auszutauschen.

Um IT-Sicherheit auch für die Zukunft zu garantieren, müssen wir ein Portfolio sicherer kryptographischer Funktionen vorbereiten. Es benötigt Verfahren, die sowohl für die Welt der allgegenwärtigen (weniger leistungsfähigen) Computer geeignet sind als auch dann sicher bleiben, wenn es leistungsfähige Quantencomputer gibt. Einige viel versprechende Kandidaten für ein solches Portfolio wurden in diesem Artikel vorgestellt; diese müssen jedoch noch sorgfältig erforscht und für die Praxis aufbereitet werden. Die Frage nach einem Verfahren zur Sicherung langfristiger Vertraulichkeit bleibt ein wichtiges offenes Problem, auf das sich zukünftige Forschung fokussieren sollte.

# Literaturverzeichnis

- [1] Charles H. Bennett and Gilles Brassard. An update on quantum cryptography. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 475–480. Springer-Verlag, 1985.
- [2] Johannes Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. CMSS – an improved Merkle signature scheme. In Rana Barua and Tanja Lange, editors, *7th International Conference on Cryptology in India - Indocrypt'06*, number 4392 in *Lecture Notes in Computer Science*, pages 349–363. Springer-Verlag, 2006.
- [3] Jan Sönke Maseberg. *Fail-Safe-Konzept für Public-Key-Infrastrukturen*. PhD thesis, TU Darmstadt, 2002.
- [4] Robert J. McEliece. A public key cryptosystem based on algebraic coding theory. *DSN progress report*, 42–44:114–116, 1978.
- [5] Ralph C. Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Department of Electrical Engineering, Stanford University, 1979.
- [6] National Institute of Standards and Technology (NIST). *Federal Information Processing Standards Publication 197: Advanced Encryption Standard*, 2002.
- [7] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [8] Peter W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
- [9] U.S. Department of Commerce, National Bureau of Standards, National Technical Information Service, Springfield, Virginia. *Federal Information Processing Standards Publication 46: Data Encryption Standard*, 1977.

# Anhang A

## Anhang

- 1 CrypTool-Menübaum
- 2 Autoren des CrypTool-Skripts
- 3 Filme und belletristische Literatur mit Bezug zur Kryptographie, Bücher mit Sammlungen einfacher Verfahren für Kinder
- 4 Lernprogramm Elementare Zahlentheorie
- 5 Kurzeinführung in das Computeralgebrasystem Sage



## A.1 CrypTool-Menübaum

Dieser Anhang enthält auf der folgenden Seite den kompletten Menübaum der CrypTool-Version 1.4.30<sup>1</sup>.

Das Haupt-Menü enthält die Service-Funktionen in den Menüs

- Datei
- Bearbeiten
- Ansicht
- Optionen
- Fenster
- Hilfe

und die eigentlichen Krypto-Funktionen in den Menüs

- Ver-/Entschlüsseln
- Digitale Signaturen/PKI
- Einzelverfahren
- Analyse.

Unter **Einzelverfahren** finden sich auch die Visualisierungen von Einzelalgorithmen und von Protokollen. Manche Verfahren sind sowohl als schnelle Durchführung (meist unter dem Menü **Ver-/Entschlüsseln**) als auch als Schritt-für-Schritt-Visualisierung implementiert.

Welche Menüeinträge gerade aktiv (also nicht ausgegraut) sind, wird durch den Typ des aktiven Dokumentfensters bestimmt: So ist z. B. die Brute-Force-Analyse für DES nur verfügbar, wenn das aktive Fenster in Hexadezimal-Darstellung geöffnet ist, während der Menüeintrag „Zufallsdaten erzeugen...“ immer verfügbar ist (auch wenn kein Dokument geöffnet ist).

---

<sup>1</sup>Parallel zu CrypTool 1.x werden im CrypTool-Projekt momentan auch die Zukunftsversionen CrypTool 2 und JCrypTool entwickelt:

- Webseite CT2: <http://www.cryptool2.vs.uni-due.de>

- Webseite JCT: <http://jcryptool.sourceforge.net>

Diese Zukunftsversionen sind zur Zeit (Dezember 2009) noch Betaversionen; sie sind aber schon stabil genug, um von Endbenutzern genutzt werden zu können. Wenn es hier Releaseversionen gibt, werden im Skript die entsprechenden Menüpfade und Menübäume ergänzt.

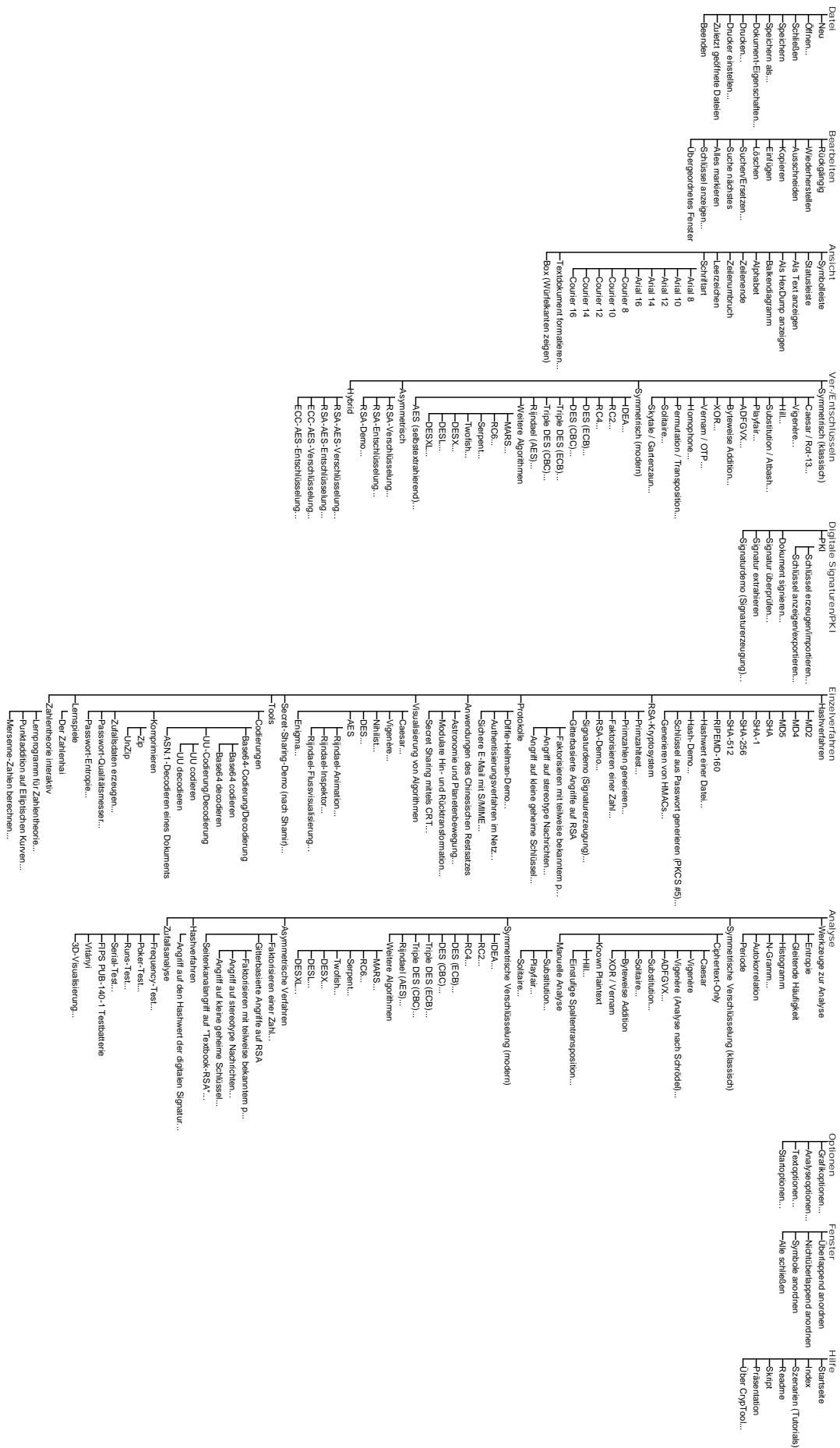


Abbildung A.1: Komplette Übersicht über den Menü-Baum von CrypTool 1.4.30

## A.2 Autoren des CrypTool-Skripts

Dieser Anhang führt die Autoren dieses Dokuments auf.

Die Autoren sind namentlich am Anfang jedes Kapitels aufgeführt, zu dem sie beigetragen haben.

**Bernhard Esslinger,**

Initiator des CrypTool-Projekts, Hauptautor dieses Skripts, Leiter IT-Sicherheit in der Deutschen Bank und Professor an der Universität Siegen.

E-Mail: [esslinger@fb5.uni-siegen.de](mailto:esslinger@fb5.uni-siegen.de).

---

**Matthias Büger,**

Mitautor des Kapitels 7 (“Elliptische Kurven”), Research Analyst bei der Deutschen Bank.

**Bartol Filipovic,**

Ursprünglicher Autor der Elliptische-Kurven-Implementierung in CrypTool und des entsprechenden Kapitels in diesem Skript.

**Henrik Koy,**

Hauptentwickler und Koordinator der CrypTool-Entwicklung seit Version 1.3, Reviewer des Skripts und T<sub>E</sub>X-Guru, Projektleiter IT und Kryptologe bei der Deutschen Bank.

**Roger Oyono,**

Implementierer des Faktorisierungs-Dialogs in CrypTool und ursprünglicher Autor des Kapitels 5 (“Die mathematischen Ideen hinter der modernen Kryptographie”).

**Jörg Cornelius Schneider,**

Design und Support von CrypTool, Kryptographie-Enthusiast und IT-Architekt und Senior-Projektleiter IT bei der Deutschen Bank.

**Christine Stötzel,**

Diplom Wirtschaftsinformatikerin an der Universität Siegen.

---

**Johannes Buchmann,**

Mitautor des Kapitels 8 (“Krypto 2020 — Perspektiven für langfristige kryptographische Sicherheit”), Vizepräsident der TU Darmstadt (TUD) und Professor an den Fachbereichen für Informatik und Mathematik der TUD. Dort hat er den Lehrstuhl für Theoretische Informatik (Kryptographie und Computer-Algebra).

**Alexander May,**

Mitautor des Kapitels 8 (“Krypto 2020 — Perspektiven für langfristige kryptographische Sicherheit”), Junior-Professor am Fachbereich Informatik der TU Darmstadt.

**Erik Dahmen,**

Mitautor des Kapitels 8 (“Krypto 2020 — Perspektiven für langfristige kryptographische Sicherheit”), Mitarbeiter am Lehrstuhl Theoretische Informatik (Kryptographie und Computeralgebra) der TU Darmstadt.

**Ulrich Vollmer,**

Mitautor des Kapitels 8 (“Krypto 2020 — Perspektiven für langfristige kryptographische Sicherheit”), Mitarbeiter am Lehrstuhl Theoretische Informatik (Kryptographie und Computeralgebra) der TU Darmstadt.

---

**Minh Van Nguyen,**

Sage-Entwickler und Reviewer des Dokuments.

## A.3 Filme und belletristische Literatur mit Bezug zur Kryptographie, Kinderbücher mit einfachen Verschlüsselungen

Kryptographische Verfahren – sowohl klassische wie moderne – fanden auch Eingang in die Literatur und in Filme. In manchen Medien werden diese nur erwähnt und sind reine Beigabe, in anderen sind sie tragend und werden genau erläutert, und manchmal ist die Rahmenhandlung nur dazu da, dieses Wissen motivierend zu transportieren. Anbei der Beginn eines Überblicks.

### A.3.1 Für Erwachsene und Jugendliche

[Poe1843] Edgar Allan Poe,

*Der Goldkäfer*, 1843.

Diese Kurzgeschichte erschien in Deutsch z.B. in der illustrierten und mit Kommentaren in den Marginalspalten versehenen Ausgabe „Der Goldkäfer und andere Erzählungen“, Gerstenbergs visuelle Weltliteratur, Gerstenberg Verlag, Hildesheim, 2002.

In dieser Kurzgeschichte beschreibt Poe als Ich-Erzähler seine Bekanntschaft mit dem sonderbaren Legrand. Mit Hilfe eines an der Küste Neuenglands gefundenen Goldkäfers, einem alten Pergament und den Dechiffrierkünsten von Legrand finden Sie den sagenhaften Schatz von Kapitän Kidd.

Die Geheimschrift besteht aus 203 kryptischen Zeichen und erweist sich als allgemeine monoalphabetische Substitutions-Chiffre (vgl. Kapitel 2.2.1). Ihre schrittweise Dechiffrierung durch semantische und syntaktische Analyse (Häufigkeiten der einzelnen Buchstaben in englischen Texten) wird ausführlich erläutert.

Der Entschlüsseler Legrand sagt darin (S. 39) den berühmten Satz: „Und es ist wohl sehr zu bezweifeln, ob menschlicher Scharfsinn ein Rätsel ersinnen kann, das menschlicher Scharfsinn bei entsprechender Hingabe nicht wieder zu lösen vermag.“

[Verne1885] Jules Verne,

*Mathias Sandorf*, 1885.

Dies ist einer der bekanntesten Romane des französischen Schriftstellers Jules Verne (1828-1905), der auch als „Vater der Science Fiction“ bezeichnet wurde.

Erzählt wird die spannende Geschichte des Freiheitskämpfers Graf Sandorf, der an die Polizei verraten wird, aber schließlich fliehen kann.

Möglich wurde der Verrat nur, weil seine Feinde eine Geheimbotschaft an ihn abfangen und entschlüsseln konnten. Dazu benötigten sie eine besondere Schablone, die sie ihm stahlen. Diese Schablone bestand aus einem quadratischen Stück Karton mit 6x6 Kästchen, wovon 1/4, also neun, ausgeschnitten waren (vgl. die Fleißner-Schablone in Kapitel 2.1.1).

[Kipling1901] Rudyard Kipling,

*Kim*, 1901.

Dieser Roman wird in der Besprechung von Rob Slade<sup>2</sup> folgendermaßen beschrieben: „Kipling packte viele Informationen und Konzepte in seine Geschichten. In „Kim“ geht es um das große „Spiel“ Spionage und Bespitzelung. Schon auf den ersten 20 Seite finden sich Authentisierung über Besitz, Denial of Service, Sich-für-jemand-anderen-Ausgeben (Impersonation), Heimlichkeit, Maskerade, Rollen-basierte Autorisierung (mit Ad-hoc-Authentisierung durch Wissen), Abhören, und Vertrauen basierend auf Datenintegrität.

---

<sup>2</sup>Siehe <http://catless.ncl.ac.uk/Risks/24.49.html#subj12>.

Später kommen noch Contingency Planning gegen Diebstahl und Kryptographie mit Schlüsselwechsel hinzu.“

Das Copyright des Buches ist abgelaufen<sup>3</sup>.

[Doyle1905] Arthur Conan Doyle,

*Die tanzenden Männchen*, 1905.

In der Sherlock-Holmes-Erzählung *Die tanzenden Männchen* (erschieden erstmals 1903 im „Strand Magazine“, und dann 1905 im Sammelband „Die Rückkehr des Sherlock Holmes“ erstmals in Buchform) wird Sherlock Holmes mit einer Geheimschrift konfrontiert, die zunächst wie eine harmlose Kinderzeichnung aussieht.

Sie erweist sich als monoalphabetische Substitutions-Chiffre (vgl. Kapitel 2.2.1) des Verbrechers Abe Slaney. Holmes knackt die Geheimschrift mittels Häufigkeitsanalyse.

[Sayers1932] Dorothy L. Sayers,

*Zur fraglichen Stunde und Der Fund in den Teufelsklippen* (Originaltitel: *Have his carcase*), Harper, 1932

(1. dt. Übersetzung *Mein Hobby: Mord* bei A. Scherz, 1964;

dann *Der Fund in den Teufelsklippen* bei Rainer Wunderlich-Verlag, 1974;

Neuübersetzung 1980 im Rowohlt-Verlag).

In diesem Roman findet die Schriftstellerin Harriet Vane eine Leiche am Strand und die Polizei hält den Tod für einen Selbstmord. Doch Harriet Vane und der elegante Amateurdetektiv Lord Peter Wimsey klären in diesem zweiten von Sayers's berühmten Harriet Vane's Geschichten den widerlichen Mord auf.

Dazu ist ein Chifftrat zu lösen. Erstaunlicherweise beschreibt der Roman nicht nur detailliert die Playfair-Chiffre, sondern auch deren Kryptoanalyse (vgl. Playfair in Kapitel 2.2.3).

[Arthur196x] Robert Arthur,

*Die 3 ????: Der geheime Schlüssel nach Alfred Hitchcock (Band 119)*, Kosmos-Verlag (ab 1960)

Darin müssen die drei Detektive Justus, Peter und Bob verdeckte und verschlüsselte Botschaften entschlüsseln, um herauszufinden, was es mit dem Spielzeug der Firma Kopferschmidt auf sich hat.

[Simmel1970] Johannes Mario Simmel,

*Und Jimmy ging zum Regenbogen*, Knaur Verlag, 1970.

Der Roman spielt zwischen 1938 und 1969 in Wien. Der Held Manuel Aranda deckt – von mehreren Geheimdiensten verfolgt – im Laufe der Handlung Stück für Stück die Vergangenheit seines ermordeten Vaters auf. Ein wichtiger Mosaikstein ist dabei ein verschlüsseltes Manuskript, das in Kapitel 33 entschlüsselt wird. Im Roman wird der Code als ein „fünfundzwanzigfacher Caesar Code“ beschrieben, tatsächlich ist es eine Vigenère-Chiffre mit einem 25 Buchstaben langen Schlüssel.

Das Buch wurde 1971 verfilmt.

---

<sup>3</sup>Sie können es lesen unter:

<http://whitewolf.newcastle.edu.au/words/authors/K/KiplingRudyard/prose/Kim/index.html>,

<http://kipling.thefreelibrary.com/Kim> oder

<http://www.readprint.com/work-935/Rudyard-Kipling>.

[Crichton1988] Michael Crichton,

*Die Gedanken des Bösen (Originaltitel: Sphere)*, Rororo, 1988.

Ein Team verschiedener Wissenschaftler wird auf den Meeresgrund geschickt, um ein 900 m langes hoch entwickeltes Raumschiff zu untersuchen. Die Eigenheiten und psychischen Probleme der Forscher treten durch lebensbedrohliche Ereignisse und ihr Abgeschnitten-sein von oben immer mehr in den Vordergrund. Es gibt viele Rätsel: Das Raumschiff liegt schon 300 Jahre da, es hat englische Beschriftungen, es führt scheinbar ein Eigenleben, die menschliche Vorstellungskraft materialisiert sich. Unter anderem erscheint auf dem Bildschirm ein im Buch vollständig abgedruckter Code, der von dem genialen Mathematiker Harry entschlüsselt werden kann: ein einfacher spiralenförmiger Ersetzungscode.

[Seed1990] Regie Paul Seed (Paul Lessac),

*Das Kartenhaus (Originaltitel: House of Cards)*, 1990 (dt. 1992).

In diesem Film versucht Ruth, hinter das Geheimnis zu kommen, das ihre Tochter verstummen ließ. Hierin unterhalten sich Autisten mit Hilfe von 5- und 6-stelligen Primzahlen (siehe Kapitel 3). Nach über eine Stunde kommen im Film die folgenden beiden (nicht entschlüsselten) Primzahlfolgen vor:

21.383, 176.081, 18.199, 113.933, 150.377, 304.523, 113.933  
193.877, 737.683, 117.881, 193.877

[Robinson1992] Regie Phil Alden Robinson,

*Sneakers - Die Lautlosen (Originaltitel: Sneakers)*, Universal Pictures Film, 1992.

In diesem Film versuchen die „Sneakers“ (Computerfreaks um ihren Boss Martin Bishop), den „Bösen“ das Dechiffrierungsprogramm SETEC abzujagen. SETEC wurde von einem genialen Mathematiker vor seinem gewaltsamen Tod erfunden und kann alle Geheimcodes dieser Welt entschlüsseln.

In dem Film wird das Verfahren nicht beschrieben<sup>4</sup>.

[Baldacci1997] David Baldacci,

*Das Labyrinth. Total Control*, Lübbe, 1997.

Jason Archer, Direktor einer Technologie-Firma, verschwindet plötzlich. Seine Frau Sidney Archer versucht, den Grund seines plötzlichen Todes herauszufinden, und entdeckt, wie das Finanzsystem missbraucht wird und dass die reale Macht bei denen mit dem meisten Geld liegt. Hier helfen dann auch gute Passworte nicht...

[Natali1997] Regie Vincenzo Natali,

*Cube (Originaltitel: Sneakers)*, Mehra Meh Film, 1997.

In diesem kanadischen Low-Budget-Film finden sich 7 sehr unterschiedliche Personen in einem endlos scheinenden Labyrinth von würfelartigen Räumen.

Die Personen wollen nach draußen, müssen dazu aber die Räume durchqueren, von denen manche tödliche Fallen darstellen. Um herauszufinden, welche Räume gefährlich sind,

---

<sup>4</sup>An dem Film hatte Leonard Adleman (das „A“ von RSA) als mathematischer Berater mitgearbeitet. Die recht lustige Geschichte über seine Mitwirkung bei Sneakers beschreibt er selbst auf seiner Homepage unter <http://www.usc.edu/dept/molecular-science/fm-sneakers.htm>. Man kann man davon ausgehen, dass es sich bei dem überall benutzten Verschlüsselungsverfahren um RSA handelt. In dem Chip ist demnach ein bis dahin unbekanntes, schnelles Faktorisierungsverfahren implementiert.

spielt Mathematik eine entscheidende Rolle: Jeder Raum hat am Eingang eine Folge von 3 mal 3 Ziffern. Zuerst nehmen sie an, dass alle Räume Fallen sind, wo wenigstens eine der 3 Zahlen eine Primzahl ist. Später stellt sich heraus, dass auch alle diejenigen Räume Fallen sind, bei denen eine der 3 Zahlen eine Potenz von genau einer Primzahl ist (Fallen sind also  $p^n$ , z.B.  $128 = 2^7$  oder  $101 = 101^1 = \text{prim}$ , aber nicht  $517 = 11 * 47$ ).

[Becker1998] Regie Harold Becker,

*Das Mercury Puzzle (Originaltitel: Mercury Rising)*, Universal Pictures Film, 1998.

Die NSA hat einen neuen Code entwickelt, der angeblich weder von Menschen noch von Computern geknackt werden kann. Um die Zuverlässigkeit zu testen, verstecken die Programmierer eine damit verschlüsselte Botschaft in einem Rätselheft.

Simon, eine neunjähriger autistischer Junge, knackt den Code. Statt den Code zu fixen, schickt ihm ein Sicherheitsbeamter einen Killer. Der FBI-Agent Art Jeffries (Bruce Willis) beschützt den Jungen und stellt den Killern eine Falle.

Das Chiffrier-Verfahren wird nicht beschrieben.

[Brown1998] Dan Brown,

*Diabolus (Originaltitel: Digital Fortress)*, Lübbe, 2005.

Dan Browns erster Roman „The Digital Fortress“ erschien 1998 als E-Book, blieb jedoch damals weitgehend erfolglos.

Die National Security Agency (NSA) hat für mehrere Milliarden US-Dollar einen gewaltigen Computer gebaut, mit dem sie in der Lage ist, auch nach modernsten Verfahren verschlüsselte Meldungen (natürlich nur die von Terroristen und Verbrechern) innerhalb weniger Minuten zu entziffern.

Ein abtrünniger Angestellter erfindet einen unbrechbaren Code und sein Computerprogramm Diabolus zwingt damit den Supercomputer zu selbstzerstörerischen Rechenoperationen. Der Plot, in dem auch die schöne Computerexpertin Susan Fletcher eine Rolle spielt, ist ziemlich vorhersehbar.

Die Idee, dass die NSA oder andere Geheimdienste jeden Code knacken können, wurde schon von mehreren Autoren behandelt: Hier hat der Supercomputer 3 Millionen Prozessoren – trotzdem ist es aus heutiger Sicht damit auch nicht annäherungsweise möglich, diese modernen Codes zu knacken.

[Elsner1999] Dr. C. Elsner,

*Der Dialog der Schwestern*, c't, Heise-Verlag, 1999.

In dieser Geschichte, die dem CrypTool-Paket als PDF-Datei beigelegt ist, unterhalten sich die Heldinnen vertraulich mit einer Variante des RSA-Verfahrens (vgl. Kapitel 4.10 ff.). Sie befinden sich in einem Irrenhaus unter ständiger Bewachung.

[Stephenson1999] Neal Stephenson,

*Cryptonomicon*, Harper, 1999.

Der sehr dicke Roman beschäftigt sich mit Kryptographie sowohl im zweiten Weltkrieg als auch in der Gegenwart. Die zwei Helden aus den 40er-Jahren sind der glänzende Mathematiker und Kryptoanalytiker Lawrence Waterhouse, und der übereifrige, morphiumsüchtige Bobby Shaftoe von den US-Marines. Sie gehören zum Sonderkommando 2702, einer Alliiertengruppe, die versucht, die gegnerischen Kommunikationscodes zu knacken und dabei ihre eigene Existenz geheim zu halten.



In der Gegenwartshandlung tun sich die Enkel der Weltkriegshelden – der Programmierreak Randy Waterhouse und die schöne Amy Shaftoe – zusammen.

Cryptonomicon ist für nicht-technische Leser teilweise schwierig zu lesen. Mehrere Seiten erklären detailliert Konzepte der Kryptographie. Stephenson legt eine ausführliche Beschreibung der Solitaire-Chiffre (siehe Kapitel 2.4) bei, ein Papier- und Bleistiftverfahren, das von Bruce Schneier entwickelt wurde und im Roman „Pontifex“ genannt wird. Ein anderer, moderner Algorithmus namens „Arethusa“ wird dagegen nicht offengelegt.

[Elsner2001] Dr. C. Elsner,

*Das Chinesische Labyrinth*, c't, Heise-Verlag, 2001.

In dieser Geschichte, die dem CrypTool-Paket als PDF-Datei beigelegt ist, muss Marco Polo in einem Wettbewerb Probleme aus der Zahlentheorie lösen, um Berater des großen Khan zu werden. Alle Lösungen sind angefügt und erläutert.

[Colfer2001] Eoin Colfer,

*Artemis Fowl*, List-Verlag, 2001.

In diesem Jugendbuch gelangt der junge Artemis, ein Genie und Meisterdieb, an eine Kopie des streng geheimen „Buches der Elfen“. Nachdem er es mit Computerhilfe entschlüsselt hat, erfährt er Dinge, die kein Mensch erfahren dürfte.

Der Code wird in dem Buch nicht genauer beschrieben.

[Howard2001] Regie Ron Howard,

*A Beautiful Mind*, 2001.

Verfilmung der von Sylvia Nasar verfassten Biographie des Spieltheoretikers John Nash. Nachdem der brillante, aber unsoziale Mathematiker geheime kryptographische Arbeiten annimmt, verwandelt sich sein Leben in einen Alptraum. Sein unwiderstehlicher Drang, Probleme zu lösen, gefährden ihn und sein Privatleben. Nash ist in seiner Vorstellungswelt ein staatstragender Codeknacker.

Konkrete Angaben zu seinen Analyseverfahren werden nicht beschrieben.

[Apted2001] Regie Michael Apted,

*Enigma – Das Geheimnis*, 2001.

Verfilmung des von Robert Harris verfassten „historischen Romans“ *Enigma* (Hutchinson, London, 1995) über die berühmteste Verschlüsselungsmaschine in der Geschichte, die in Bletchley Park nach polnischen Vorarbeiten gebrochen wurde. Die Geschichte spielt 1943, als der eigentliche Erfinder Alan Turing schon in Amerika war. So kann der Mathematiker Tom Jericho als Hauptperson in einem spannenden Spionagethriller brillieren.

Konkrete Angaben zu dem Analyseverfahren werden nicht gemacht.

[Isau1997] Ralf Isau,

*Das Museum der gestohlenen Erinnerungen*, Thienemann-Verlag, 1997/2003.

Ein sehr spannender, hervorragend recherchierter und doch leicht zu lesender Roman mit einem tiefen Hintersinn.

Als die Zwillinge Oliver und Jessica von ihren Ferien zurückkommen, haben sie ihren Vater vergessen. Die Realität verschiebt sich und niemand scheint es zu bemerken. An einigen Stellen bleiben manchmal Spuren zurück, die man entziffern kann. Zentrum der

Geschichte ist das Ishtar-Tor im Berliner Pergamon-Museum. Nur mit dem Scharfsinn einer irischen Professorin (die gleichzeitig Computerexpertin, Archäologin und Philologin ist), den besonderen Beziehungen zwischen Zwillingen und den vereinten Kräften der Computergemeinschaft kann der letzte Teil des Spruches gelöst werden. Das Buch wurde als bestes Jugendbuch ausgezeichnet und liegt in 8 Sprachen vor.

[Brown2003] Dan Brown,

*Sakrileg (Originaltitel: The Da Vinci Code)*, Lübbe, 2004.

Der Direktor des Louvre wird in seinem Museum vor einem Gemälde Leonardos ermordet aufgefunden, und der Symbolforscher Robert Langdon gerät in eine Verschwörung.

Innerhalb der Handlung werden verschiedene klassische Chiffren (Substitution wie z.B. Caesar oder Vigenere, sowie Transposition und Zahlencodes) angesprochen. Außerdem klingen interessante Nebenbemerkungen über Schneier oder die Sonnenblume an. Der zweite Teil des Buches ist sehr von theologischen Betrachtungen geprägt.

Das Buch ist einer der erfolgreichsten Romane der Welt.

[McBain2004] Scott McBain,

*Der Mastercode (Originaltitel: Final Solution)*, Knaur, 2005.

In einer nahen Zukunft haben Politiker, Militärs und Geheimdienstchefs aus allen Staaten in korrupter Weise die Macht übernommen. Mit einem gigantischen Computernetzwerk names „Mother“ und vollständiger Überwachung wollen sie die Machtverteilung und Kommerzialisierung für immer festschreiben. Menschen werden ausschließlich nach ihrem Kreditrating bewertet und global agierende Unternehmen entziehen sich jeder demokratischen Kontrolle. Innerhalb des Thrillers wird die offensichtliche Ungerechtigkeit, aber auch die realistische Möglichkeit dieser Entwicklung immer wieder neu betont.

In den Supercomputer „Mother“ wurde m.H. eines Kryptologen ein Code zur Deaktivierung eingebaut: In einem Wettrennen mit der Zeit versuchen Lars Pedersen, Oswald Plevy, die amerikanische Präsidentin, der britische Regierungschef und eine unbekannte Finnin namens Pia, die den Tod ihres Bruders rächen will, den Code zur Deaktivierung zu starten. Auf der Gegenseite agiert eine Gruppe mörderischer Verschwörer unter Führung des britischen Außenministers und des CIA-Chefs.

Die englische Originalfassung „The Final Solution“ wurde als Manuskript an Harper Collins, London verkauft, ist dort aber nicht erschienen.

[Burger2006] Wolfgang Burger,

*Heidelberger Lügen*, Piper, 2006.

In diesem Kriminalroman mit vielen oft unabhängigen Handlungssträngen und lokalen Geschichten geht es vor allem um den Kriminalrat Gerlach aus Heidelberg. Auf S. 207 f. wird aber auch der kryptologische Bezug von einem der Handlungsstränge kurz erläutert: der Soldat Hörle hatte Schaltpläne eines neuen digitalen NATO-Entschlüsselungsgerätes kopiert und der Ermordete hatte versucht, seine Erkenntnisse an die Chinesen zu verkaufen.

[Vidal2006] Agustin Sanchez Vidal,

*Kryptum*, Dtv, 2006.

Der erste Roman des spanischen Professors der Kunstgeschichte ähnelt Dan Browns „Sakrileg“ aus dem Jahre 2003, aber angeblich hat Vidal schon 1996 begonnen, daran zu

schreiben. Vidals Roman ist zwischen historischem Abenteuerroman und Mystery-Thriller angesiedelt und war in Spanien ein Riesenerfolg.

Im Jahre 1582 wartet Raimundo Randa, der sein Leben lang einem Geheimnis auf der Spur war, im Alkazar auf seinen Inquisitionsprozess. Dieses Geheimnis rankt sich um ein mit kryptischen Zeichen beschriftetes Pergament, von dem eine mysteriöse Macht ausgeht. Rund 400 Jahre später kann sich die amerikanische Wissenschaftlerin Sara Toledano dieser Macht nicht entziehen, bis sie in Antigua verschwindet. Ihr Kollege, der Kryptologe David Calderon, und ihre Tochter Rachel machen sich auf die Suche nach ihr und versuchen gleichzeitig, den Code zu knacken. Aber auch Geheimorganisationen wie die NSA sind hinter dem Geheimnis des „letzten Schlüssels“ her. Sie sind bereit, dafür über Leichen zu gehen.

[Larsson2007] Stieg Larsson,

*Verdammnis (Originaltitel: Flickan som lekte med elden)*, Heyne, 2007.

Der Autor wurde 2006 postum mit dem Skandinavischen Krimipreis als bester Krimiautor Skandinaviens geehrt. Die Superheldin Lisbeth Salander nutzt PGP und beschäftigt sich nebenbei auch mit mathematischen Rätseln wie dem Satz von Fermat.

[Schröder2008] Rainer M. Schröder,

*Die Judas-Papiere*, Arena, 2008.

„Historienthriller“: Lord Pembroke hat im Jahre 1899 drei Männer und eine Frau in der Hand und beauftragt sie, die verschlüsselten Botschaften in dem Notizbuch seines verstorbenen Bruders Mortimer zu entschlüsseln und das Judas-Evangelium zu finden, das das Ende der Christenheit einläuten könnte. Dazu müssen sie Rätsel an vielen Orten der Welt lösen. Im Buch finden sich klassische Verfahren wie Polybius (S. 195) oder die Freimaurer-Chiffre (S. 557).

[Hill2009] Tobias Hill,

*Der Kryptograph*, C. Bertelsmann, 2009.

London 2021: Die Firma SoftMark hat eine elektronische Währung entwickelt und etabliert, die durch einen nicht entschlüsselbaren Code allen Nutzern höchste Sicherheit garantiert. Der Erfinder und Firmengründer John Law, wegen seiner mathematischen Begabung auch der Kryptograph genannt, ist damit zum reichsten Mann der Welt geworden. Doch dann wird der Code geknackt, und in einer dadurch verursachten Weltwirtschaftskrise geht auch die Firma von John Law pleite. Außerdem wird die Steuerfahnderin Anna Moore auf ihn angesetzt.

[Eschbach2009] Andreas Eschbach,

*Ein König für Deutschland*, Lübbe, 2009.

Der Roman dreht sich um die Manipulierbarkeit von Wahlcomputern.

Vincent Merrit, ein junger US-amerikanischer Programmierer, wird erpresst, ein solches Programm zu schreiben. Neben kommerziell orientierten Erpressern kommen z.B. auch Online-Rollenspiele und Live-Rollenspiele (LARPs) in dem Roman vor. Weil Merrit den Missbrauch seines Programms ahnte, baute er eine Hintertür ein: Nimmt eine Partei namens VWM an der Wahl teil, erhält sie automatisch 95 % der Stimmen ...

Die fiktive Handlung des Romans beruht auf zahlreichen überprüfbaren und genau recherchierten Tatsachen, auf die in Fußnoten hingewiesen wird.

Während die kryptographischen Protokolle sicher gemacht werden können, bleiben ihre Implementierung und ihre Organisation anfällig gegen Missbrauch.

**Anmerkung 1:** Weitere Beispiele für Kryptologie in der belletristischen Literatur finden sich z.B. auf der folgenden Webseite:

[http://www.staff.uni-mainz.de/pommeren/Kryptologie99/Klassisch/1\\_Monoalph/Literat.html](http://www.staff.uni-mainz.de/pommeren/Kryptologie99/Klassisch/1_Monoalph/Literat.html)

Für die ältere Literatur (z.B. von Jules Verne, Karl May, Arthur Conan Doyle, Edgar Allen Poe) sind darauf sogar die Links zu den eigentlichen Textstellen enthalten.

**Anmerkung 2:** Abgebildet und beschrieben finden Sie einige dieser Buchtitelseiten auf der Webseite von Tobias Schrödel, der sich vor allem mit antiken Büchern zur Kryptographie beschäftigt:

[http://tobiasschroedel.com/crypto\\_books.php](http://tobiasschroedel.com/crypto_books.php)

**Anmerkung 3:** Wenn Sie weitere Literatur und Filme wissen, wo Kryptographie eine wesentliche Rolle spielt, dann würden wir uns sehr freuen, wenn Sie uns den genauen Titel und eine kurze Erläuterung zum Film-/Buchinhalt zusenden würden. Herzlichen Dank.

### A.3.2 Für Kinder und Jugendliche

Die folgende Auflistung enthält Filme und „Kinderbücher“. Die Kinderbücher hier enthalten Sammlungen von einfachen kryptographischen Verfahren, didaktisch und spannend aufbereitet:

[**Mosesxxxx**] [Ohne Autorenangabe],

*Streng geheim – Das Buch für Detektive und Agenten*, Edition moses, [ohne Angabe der Jahreszahl].

Ein dünnes Buch für kleinere Kinder mit Inspektor Fox und Dr. Chicken.

[**Para1988**] Para,

*Geheimschriften*, Ravensburger Taschenbuch Verlag, 1988 (erste Auflage 1977).

Auf 125 eng beschriebenen Seiten werden in dem klein-formatigen Buch viele verschiedene Verfahren erläutert, die Kinder selbst anwenden können, um ihre Nachrichten zu verschlüsseln oder unlesbar zu machen. Ein kleines Fachwörter-Verzeichnis und eine kurze Geschichte der Geheimschriften runden das Büchlein ab.

Gleich auf S. 6 steht in einem old-fashioned Style für den Anfänger „Das Wichtigste zuerst“ über Papier- und Bleistiftverfahren (vergleiche Kapitel 2): „Wollte man Goldene Regeln für Geheimschriften aufstellen, so könnten sie lauten:

- Deine geheimen Botschaften müssen sich an jedem Ort zu jeder Zeit mit einfachsten Mitteln bei geringstem Aufwand sofort anfertigen lassen.
- Deine Geheimschrift muss für Deine Partner leicht zu merken und zu lesen sein. Fremde hingegen sollen sie nicht entziffern können.  
Merke: Schnelligkeit geht vor Raffinesse, Sicherheit vor Sorglosigkeit.
- Deine Botschaft muss immer knapp und präzise sein wie ein Telegramm. Kürze geht vor Grammatik und Rechtschreibung. Alles Überflüssige weglassen (Anrede, Satzzeichen). Möglichst immer nur Groß- oder immer nur Kleinbuchstaben verwenden.“

[**Müller-Michaelis2002**] Matthias Müller-Michaelis,

*Das Handbuch für Detektive. Alles über Geheimsprachen, Codes, Spurenlesen und die großen Detektive dieser Welt*, Südwest-Verlag, 2002.

Broschiert, 96 Seiten.

[**Kippenhahn2002**] Rudolf Kippenhahn,

*Streng geheim! – Wie man Botschaften verschlüsselt und Zahlencodes knackt*, rororo, 2002.

In dieser Geschichte bringt ein Großvater, ein Geheimschriftexperte, seinen 4 Enkeln und deren Freunden bei, wie man Botschaften verschlüsselt, die niemand lesen soll. Da es jemand gibt, der die Geheimnisse knackt, muss der Großvater den Kindern immer komplizierte Verfahren beibringen.

In dieser puren Rahmenhandlung werden die wichtigsten klassischen Verfahren und ihre Analyse kindgerecht und spannend erläutert.

[**Harder2003**] Corinna Harder und Jens Schumacher,

*Streng geheim. Das große Buch der Detektive*, Moses, 2003.

Gebundene Ausgabe: 118 Seiten.

[**Talke-Baisch2003**] Helga Talke und Milena Baisch,  
*Dein Auftrag in der unheimlichen Villa. Kennwort Rätselkrimi*, Loewe, 2003.  
Ab Klasse 4, <http://www.antolin.de>  
Junge Rätseldetektive lösen bei ihren Einsätzen auch einfache Geheimsprachen und Codes.

[**Flessner2004**] Bernd Flessner,  
*Die 3 ????: Handbuch Geheimbotschaften*, Kosmos, 2004.  
Auf 127 Seiten wird sehr verständlich, spannend, und strukturiert nach Verfahrenstyp  
geschildert, welche Geheimsprachen (Navajo-Indianer oder Dialekte) und welche Geheim-  
schriften (echte Verschlüsselung, aber auch technische und linguistische Steganographie)  
es gab und wie man einfache Verfahren entschlüsseln kann.  
Bei jedem Verfahren steht, wenn es in der Geschichte oder in Geschichten Verwendung  
fand [wie in Edgar Allan Poe's „Der Goldkäfer“, wie von Jules Verne's Held Mathias San-  
dorf oder von Astrid Lindgren's Meisterdetektiv Kalle Blomquist, der die ROR-Sprache  
verwendet (ähnliche Einfüge-Chiffren sind auch die Löffel- oder die B-Sprache)].  
Dieses Buch ist ein didaktisch hervorragender Einstieg für jüngere Jugendliche.

[**Zübert2005**] Regie Christian Zübert,  
*Der Schatz der weißen Falken*, 2005.  
Dieser spannende Kinder-Abenteuer-Film knüpft an die Tradition von Klassikern wie  
„Tom Sawyer und Huckleberry Finn“ oder Enid Blytons „Fünf Freunde“ an. Er spielt  
im Sommer des Jahres 1981. In einer halb verfallenen Villa finden 3 Jugendliche die  
Schatzkarte der „Weißen Falken“, die sie mit Hilfe eines Computer entschlüsseln. Verfolgt  
von einer anderen Jugendbande machen sie sich auf zu einer alten Burgruine.

**Anmerkung 1:** Abgebildet und beschrieben finden Sie viele dieser Kinderbuch-Titelseiten auf  
der Webseite von Tobias Schrödel, der sich vor allem mit antiken Büchern zur Kryptographie  
beschäftigt:

[http://tobiasschroedel.com/crypto\\_books.php](http://tobiasschroedel.com/crypto_books.php)

**Anmerkung 2:** Wenn Sie weitere Bücher kennen, die Kryptographie didaktisch kindergerecht  
aufbereiten, dann würden wir uns sehr freuen, wenn Sie uns den genauen Buchtitel und eine  
kurze Erläuterung zum Buchinhalt zusenden würden. Herzlichen Dank.

## A.4 Lernprogramm Elementare Zahlentheorie

In CrypTool ist ein interaktives Lernprogramm zur elementaren Zahlentheorie, genannt „ZT“, enthalten.<sup>5</sup>

Das Lernprogramm „NT“ (Zahlentheorie) von Martin Ramberger führt in die Zahlentheorie ein und visualisiert viele der Verfahren und Konzepte. Wo nötig zeigt es auch die entsprechenden mathematischen Formeln. Oftmals können die mathematischen Verfahren dynamisch mit eigenen kleinen Zahlenbeispielen ausprobiert werden.

Die Inhalte basieren vor allem auf den Büchern von [Buchmann2004] und [Scheid2003].

Dieses visualisierte Lernprogramm wurde mit Authorware 4 erstellt.

**Bitte um Erweiterung/Upgrade:** Ein Update auf die neueste Version von Authorware oder auf eine andere Entwicklungsplattform wäre wünschenswert. Wenn sich hierzu Interessenten finden, würde ich mich sehr freuen (bitte E-Mail an den Autor des CrypTool-Skriptes).

**Abbildungen:** Die Abbildungen A.2 bis A.9 vermitteln einen Eindruck des Lernprogramms „NT“:

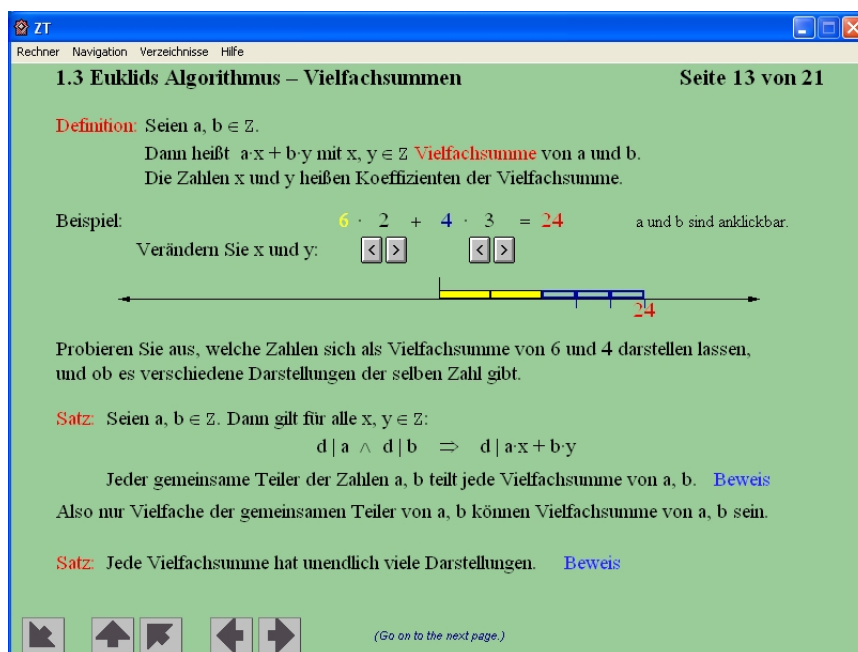


Abbildung A.2: Jeder gemeinsame Teiler zweier Zahlen teilt auch alle ihre Linearkombinationen

<sup>5</sup>ZT können Sie in CrypTool über das Menü **Einzelverfahren \ Zahlentheorie interaktiv \ Lernprogramm für Zahlentheorie** aufrufen.

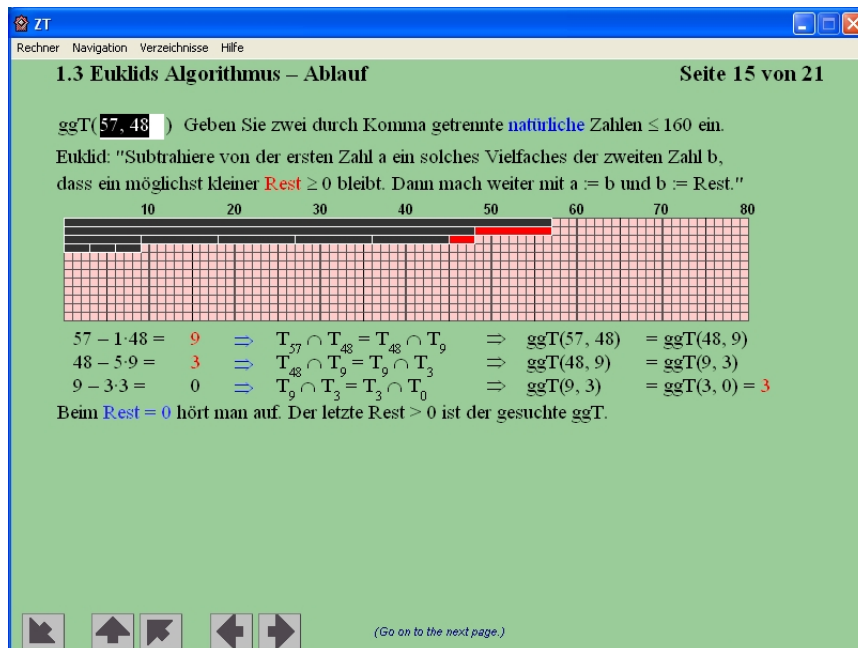


Abbildung A.3: Euklids Algorithmus zur Bestimmung des ggT

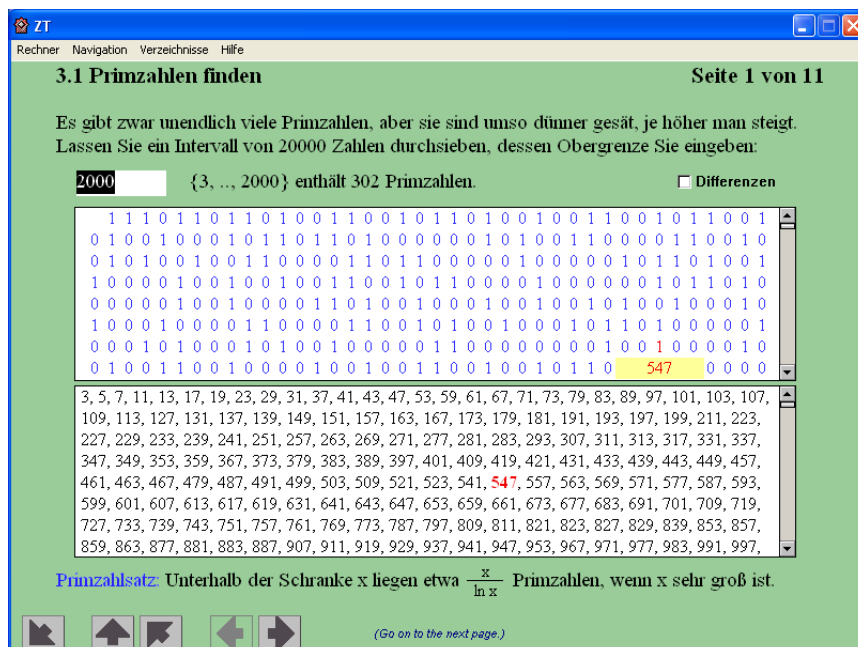


Abbildung A.4: Verteilung der Primzahlen und ihrer Differenzen



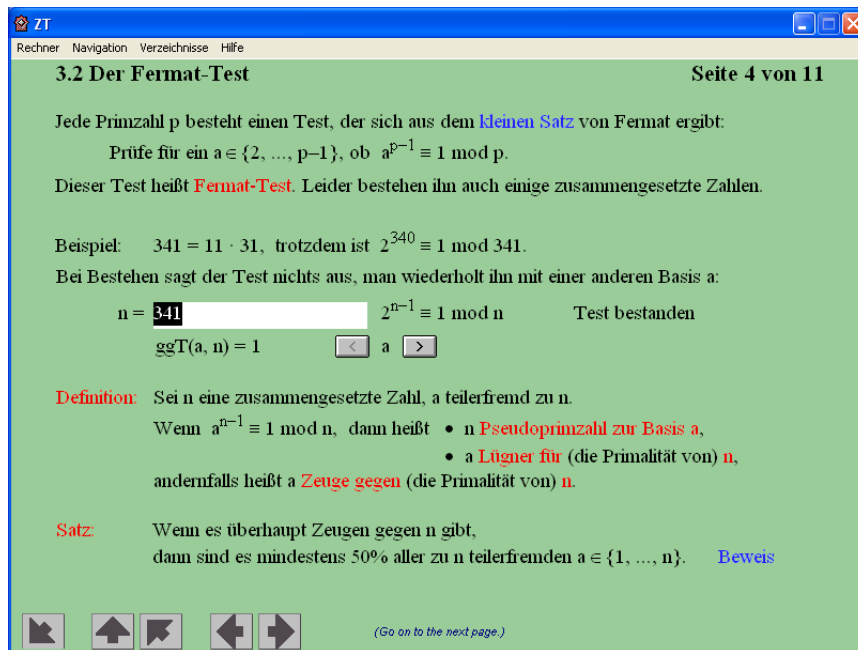


Abbildung A.5: Primzahlen finden mit dem Primzahltest nach Fermat

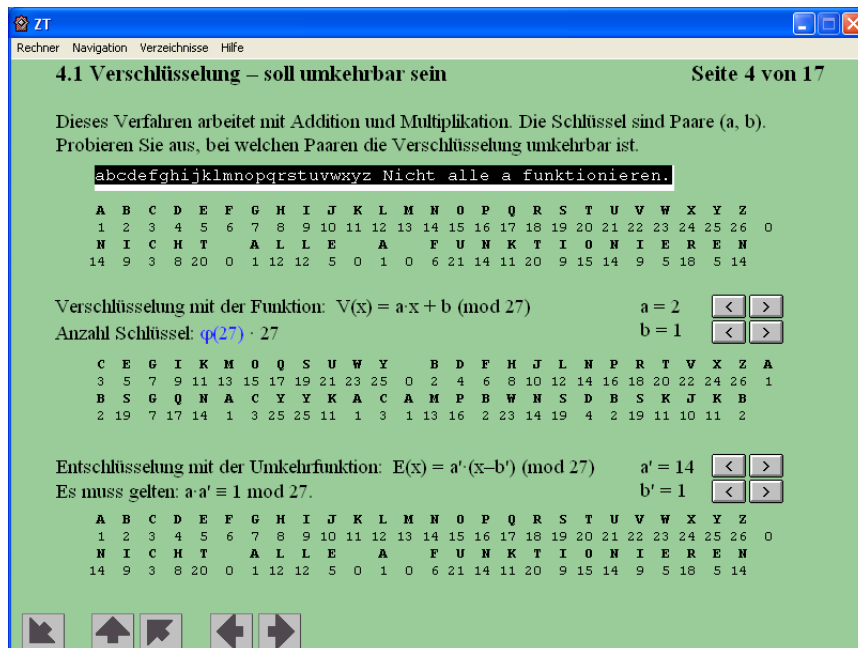


Abbildung A.6: Umkehrbarkeit von Verschlüsselungsalgorithmen am Beispiel additiver Chiffren

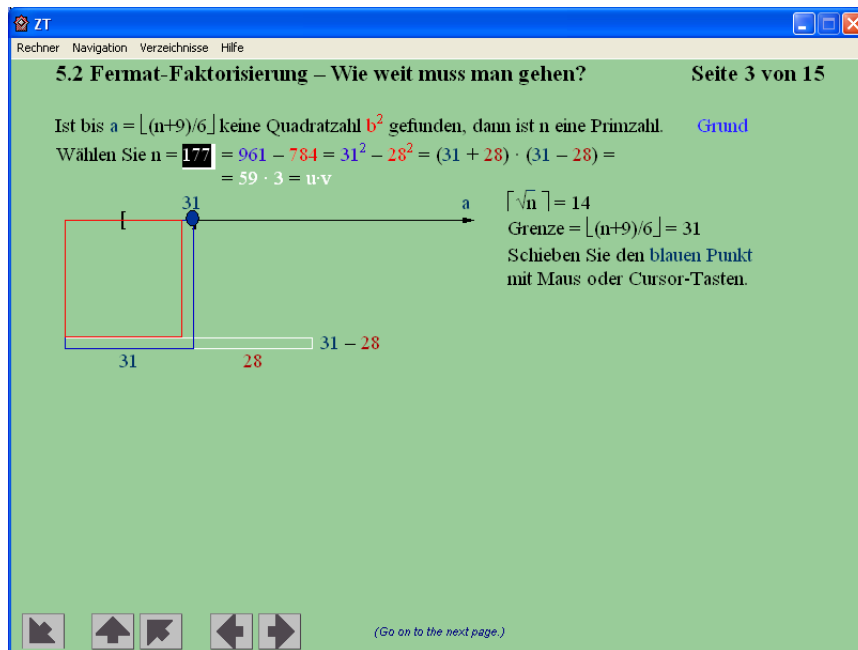


Abbildung A.7: Fermat-Faktorisierung m.H. der 3. Binomischen Formel

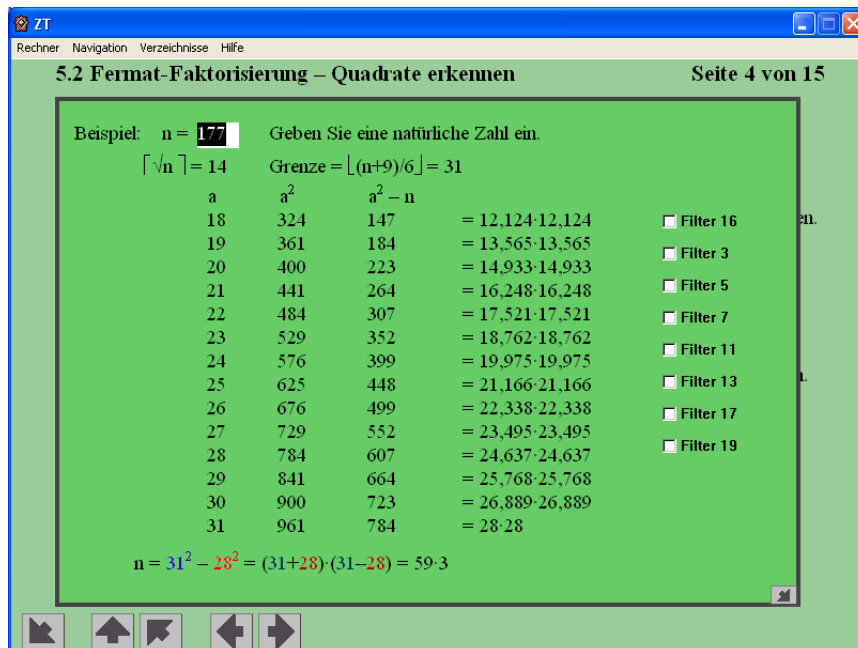


Abbildung A.8: Fermat-Faktorisierung: Quadrate erkennen

Rechner
Navigation
Verzeichnisse
Hilfe

### 5.3 Pollards Rho-Faktorisierung – Kreisfindungs-Algorithmen

Seite 8 von 15

Als Pollard seinen  $\rho$ -Algorithmus 1975 vorstellte, benutzte er den Kreisfindungs-Algorithmus von Floyd:

Floyd startet zwei Variablen  $x$  und  $y$  mit dem selben Wert und rechnet in jedem Schritt:

$$x_{i+1} := f(x_i), \quad x_{i+2} := f(x_{i+1})$$

$$y_{j+1} := f(y_j)$$

Also durchläuft  $x$  die Folge doppelt so schnell wie  $y$  und überbrundet  $y$  bald. Dann ist der Kreis gefunden.

1980 stellte Brent einen Kreisfindungs-Algorithmus vor, der in jedem Schritt nur einmal rechnet, nämlich so:

Wenn  $i$  eine 2er-Potenz ist, setze  $y := x_i$ .

Berechne  $x_{i+1} := f(x_i)$  und vergleiche den aktuellen Wert  $x_i$  mit  $y$ .

Brent-Illustration mit Zahlen

Das Bild verewigte Pollard im Namen  $\rho$ -Algorithmus.

Geben Sie für die Länge von Vorperiode und Periode zwei Zahlen  $< 1000$  ein, die erste  $\geq 0$ , die zweite  $> 0$ :

12 31

(Go on to the next page.)

Abbildung A.9: Pollards Rho-Faktorisierung: Kreisfindungsalgorithmus nach Floyd

# Literaturverzeichnis

- [Buchmann2004] Johannes Buchmann,  
*Einführung in die Kryptographie*, Springer, 3. Auflage, 2004.
- [Scheid2003] Harald Scheid,  
*Zahlentheorie*, Spektrum Akademischer Verlag, 3. Auflage, 2003.

## A.5 Kurzeinführung in das Computeralgebrasystem Sage

Dieses Skript enthält zahlreiche mit Sage erstellte Programmbeispiele. Sage ist ein Open-Source Computer-Algebra-System (CAS), das für Lehre, Studium und Forschung eingesetzt wird. Sage kombiniert viele hochwertige Open-Source-Pakete<sup>6</sup> und liefert den Zugang zu deren Funktionalität über ein gemeinsames, auf der Programmiersprache Python basierendes Interface<sup>7</sup>.

Sage kann man wie andere CAS auch auf vielfältige Weise nutzen: als mächtigen Taschenrechner; als Tool, das Studenten beim Mathematikstudium hilft; oder als Programmier-Umgebung, um Algorithmen zu prototypen oder um Forschung im Bereich der algorithmischen Aspekte der Mathematik zu betreiben.

Einen schnellen Eindruck von Sage erhält man mit der „Einladung zu Sage“ von David Joyner<sup>8</sup>.

Die offizielle Sage Online-Dokumentation<sup>9</sup> ist verfügbar unter: <http://www.sagemath.org/doc>.

Es gibt inzwischen viele PDF- und HTML-Dokumente über Sage, so dass wir als guten Startpunkt nur einige wenige nennen<sup>10</sup>.

Auch beim Studium der Kryptologie können fertige Sage-Module genutzt werden<sup>11</sup>.

Ein fertiger Kryptographie-Kurs von David Kohel, der Sage nutzt, findet sich unter

<http://www.sagemath.org/library/crypto.pdf>

bzw. derselbe Kurs in einer eventuell neueren Fassung

<http://sage.math.washington.edu/home/wdj/teaching/kohel-crypto.pdf> .

## Sage-Benutzerschnittstellen

Sage ist kostenlos und kann von folgender Webseite herunter geladen werden:

<http://www.sagemath.org>

Standardmäßig nutzt man die Sage-Kommandozeile als Interface, wie im folgenden Bild A.10 zu sehen ist. Es gibt jedoch auch ein grafisches Benutzerinterface für diese Software in Form des Sage-Notebooks (siehe Bild A.11). Und schließlich kann man Sage-Notebooks<sup>12</sup> auch online auf verschiedenen Servern nutzen, ohne Sage lokal zu installieren, z.B.:

---

<sup>6</sup>Einen Eindruck von der Größe von Sage erhält man, wenn man es selbst compiliert: Die heruntergeladenen Sourcen von Sage 4.1 brauchten zur Compilierung auf einem durchschnittlichen Linux-PC rund 5 h (inklusive aller Bibliotheken). Danach nahm es 1,8 GB Plattenplatz ein.

<sup>7</sup>Es gibt auch ein relativ einfaches Interface für die Sprache C, genannt Cython, mit der man eigene Funktionen in Sage stark beschleunigen kann.

Siehe [http://openwetware.org/wiki/Open\\_writing\\_projects/Sage\\_and\\_cython\\_a\\_brief\\_introduction](http://openwetware.org/wiki/Open_writing_projects/Sage_and_cython_a_brief_introduction).

<sup>8</sup><http://sage.math.washington.edu/home/wdj/teaching/calcl-sage/an-invitation-to-sage.pdf>  
(Letztes Update 2009).

<sup>9</sup>Die entsprechenden offiziellen PDF-Dokumente können herunter geladen werden von  
<http://www.sagemath.org/help.html>

<sup>10</sup>- „Bibliothek“: <http://www.sagemath.org/library/index.html>,  
- „Dokumentationsprojekt“: <http://wiki.sagemath.org/DocumentationProject>,  
- „Lehrmaterial“: [http://wiki.sagemath.org/Teaching\\_with\\_SAGE](http://wiki.sagemath.org/Teaching_with_SAGE).

<sup>11</sup>- Sourcen der Module im Verzeichnis `SAGE_ROOT/devel/sage-main/sage/crypto`.

- Überblick, welche Kryptographie momentan in Sage enthalten ist:

<http://www.sagemath.org/doc/reference/sage/crypto/>

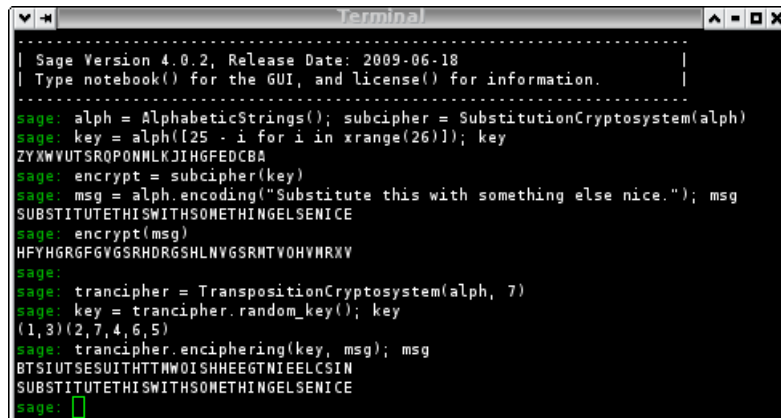
- Diskussionen über Lernaspekte beim Entwickeln weiterer Krypto-Module in Sage:

[http://groups.google.com/group/sage-devel/browse\\_thread/thread/c5572c4d8d42d081](http://groups.google.com/group/sage-devel/browse_thread/thread/c5572c4d8d42d081)

<sup>12</sup>Weitere Details zu Sage-Notebooks finden Sie in Kapitel 7.9.2 („Implementierung Elliptischer Kurven zu Lehrzwecken“ ⇒ „Sage“)

<http://www.sagenb.org> oder  
<http://sage.mathematik.uni-siegen.de:8000>

Sage läuft unter den Betriebssystemen Linux, Mac OS X und Windows. Auf der Windows-Plattform läuft die komplette Sage-Distribution momentan nur als a VMware-Image. Allerdings wird gerade an einem vollen, native Port von Sage auf Windows gearbeitet.



```
Sage Version 4.0.2, Release Date: 2009-06-18
Type notebook() for the GUI, and license() for information.

sage: alph = AlphabeticStrings(); subcipher = SubstitutionCryptosystem(alph)
sage: key = alph([25 - i for i in xrange(26)]); key
ZYXWVUTSRQPONMLKJIHGFEDCBA
sage: encrypt = subcipher(key)
sage: msg = alph.encoding("Substitute this with something else nice."); msg
SUBSTITUTETHISWITHSOMETHINGELSENICE
sage: encrypt(msg)
HFYHGRGFGVGSRRHDSHLNVGSRNTVOHVMRXV
sage:
sage: trancipher = TranspositionCryptosystem(alph, 7)
sage: key = trancipher.random_key(); key
(1,3)(2,7,4,6,5)
sage: trancipher.enciphering(key, msg); msg
BTSTUTSESUITHTTMMOISHHEEGTNIIEELCSIN
SUBSTITUTETHISWITHSOMETHINGELSENICE
sage: 
```

Abbildung A.10: Sage-Kommandozeilen-Interface



Abbildung A.11: Sage-Notebook-Interface<sup>13</sup>

## Hilfe beim Benutzen von Sage

Wenn man Sage auf der Kommandozeile startet, erhält etwas wie die folgenden Zeilen:

```
mnemonic:~$ sage
-----
| Sage Version 4.1, Release Date: 2009-07-09                |
| Type notebook() for the GUI, and license() for information. |
-----

sage: help
Type help() for interactive help, or help(object) for help about object.
sage:
sage:
sage: help()

Welcome to Python 2.6! This is the online help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics". Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".
```

Viele weitere Hilfen gibt es als offizielle Sage-Dokumentation, die mit jedem Release von Sage verteilt wird (siehe Bild A.12). Zur offiziellen Sage-Standard-Dokumentation gehören folgende Dokumente:

- Tutorial — Das Tutorial ist für Sage-Einsteiger. Es ist dafür gedacht, sich in ein bis drei Stunden mit den wichtigsten Funktionen vertraut zu machen.
- Constructions — Dieses Dokument ist im Stil eines „Kochbuchs“ mit einer Sammlung von Antworten auf Fragen zur Konstruktion von Sage-Objekten.
- Developers' Guide — Dieser Führer ist für Entwickler, die selbst Sage mit weiter entwickeln wollen. Enthalten sind darin z.B. Hinweise zum Stil und zu Konventionen beim Programmieren, zur Modifikation von Sage-Kern-Bibliotheken oder von Sage-Standard-Dokumentation, und zum Code-Review und zur Software-Verteilung.
- Reference Manual — Dieses Handbuch enthält die komplette Dokumentation aller wichtigen Sage-Funktionen. Zu jeder Klassen-Beschreibung gibt es mehrere Code-Beispiele. Alle Code-Beispiele im Referenz-Handbuch werden bei jedem neuen Sage-Release getestet.
- Installation Guide — Dieser Führer erklärt, wie man Sage auf verschiedenen Plattformen installiert.
- A Tour of Sage — Diese Tour durch Sage zeigt exemplarisch verschiedene Funktionen, die für Einsteiger sinnvoll sind.

---

<sup>13</sup>Um das grafische Sage-Interface lokal zu starten, muss man auf der Sage Kommandozeile `notebook()` eingeben. Danach startet der eingestellte Browser (Iceweasel, Firefox, IE, ...) z.B. mit der URL <http://localhost:8000>.

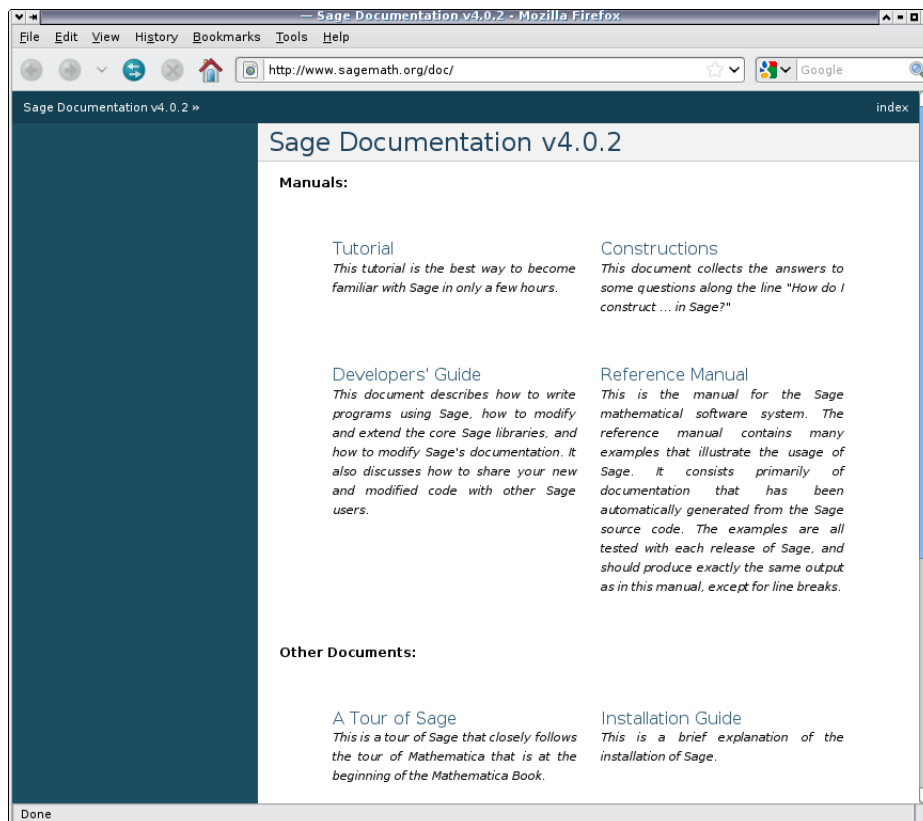


Abbildung A.12: Die Standard-Dokumentation von Sage

- Numerical Sage — Dieses Dokument führt Werkzeuge auf, die in Sage für numerische Mathematik verfügbar sind.
- Three Lectures about Explicit Methods in Number Theory Using Sage — Drei Vorlesungen über Methoden der Zahlentheorie, die explizit Sage nutzen. Dieses Dokument zeigt wie man mit Sage Berechnungen in fortgeschrittener Zahlentheorie durchführt.

Von der Sage-Kommandozeile erhält man eine Liste aller verfügbaren Kommandos (Funktionsnamen etc.), die ein bestimmtes Muster haben, wenn man die ersten Zeichen tippt, und dann die „Tab“-Taste drückt:

```
sage: Su[TAB]
Subsets                Subwords                SuzukiGroup
SubstitutionCryptosystem SupersingularModule
```

Wenn man den genauen Namen eines Kommandos kennt, kann man die `help`-Funktion nutzen oder das Fragezeichen „?“ anfügen, um weitere Informationen zu diesem Kommando zu erhalten. Zum Beispiel liefert das Kommando `help(SubstitutionCryptosystem)` die Dokumentation zur eingebauten Klasse `SubstitutionCryptosystem`. Mit dem Fragezeichen erhalten wir die Dokumentation zu dieser Klasse auf folgende Weise:

```
sage: SubstitutionCryptosystem?
Type: type
Base Class: <type 'type'>
String Form: <class 'sage.crypto.classical.SubstitutionCryptosystem'>
```



Namespace:Interactive  
File:/home/mvngu/usr/bin/sage-3.4.1/local/lib/python2.5/site-packages/sage/crypto/classical.py  
Docstring:

Create a substitution cryptosystem.

INPUT:

- ‘S’ - a string monoid over some alphabet

OUTPUT:

- A substitution cryptosystem over the alphabet ‘S’.

EXAMPLES::

```
sage: M = AlphabeticStrings()
sage: E = SubstitutionCryptosystem(M)
sage: E
Substitution cryptosystem on Free alphabetic string monoid
on A-Z
sage: K = M([ 25-i for i in range(26) ])
sage: K
ZYXWVUTSRQPONMLKJIHGFEDCBA
sage: e = E(K)
sage: m = M(‘THECATINTHEHAT’)
sage: e(m)
GSVXZGRMGSVSZG
```

TESTS::

```
sage: M = AlphabeticStrings()
sage: E = SubstitutionCryptosystem(M)
sage: E == loads(dumps(E))
True
```

Weitere Unterstützung für spezifische Probleme gibt es in den Archiven der **sage-support** Mailing-Liste unter

<http://groups.google.com/group/sage-support>

## Beispiele für in Sage eingebaute mathematische Funktionen

Hier sind ein paar kleine Beispiele<sup>14</sup> (alle für das Kommandozeilen-Interface – zur einfacheren Nutzung), um zu sehen, was man mit Sage machen kann:

---

### Sage-Beispiel A.1 Einige kleine Beispiele in Sage aus verschiedenen Gebieten der Mathematik

---

```
# * Analysis (Infinitesimalrechnung):
sage: x=var('x')
sage: p=diff(exp(x^2),x,10)*exp(-x^2)
sage: p.simplify_exp()
1024 x^10 + 23040 x^8 + 161280 x^6 + 403200 x^4 + 302400 x^2 + 30240

# * Lineare Algebra:
sage: M=matrix([[1,2,3],[4,5,6],[7,8,10]])
sage: c=random_matrix(ZZ,3,1);c
[ 7 ]
[-2 ]
[-2 ]
sage: b=M*c
sage: M^-1*b
[ 7 ]
[-2 ]
[-2 ]

# * Zahlentheorie:
sage: p=next_prime(randint(2^49,2^50));p
1022095718672689
sage: r=primitive_root(p);r
7
sage: pl=log(mod(10^15,p),r);pl
1004868498084144
sage: mod(r,p)^pl
1000000000000000

# * Endliche Körper (\url{http://de.wikipedia.org/wiki/Endlicher_K%C3%B6rper}):
sage: F.<x>=GF(2) []
sage: G.<a>=GF(2^4,name='a',modulus=x^4+x+1)
sage: a^2/(a^2+1)
a^3 + a
sage: a^100
a^2 + a + 1
sage: log(a^2,a^3+1)
13
sage: (a^3+1)^13
a^2
```

---

<sup>14</sup>Diese Beispiele stammen aus dem Blog von Dr. Alasdair McAndrew, Victoria University,  
<http://amca01.wordpress.com/2008/12/19/sage-an-open-source-mathematics-software-system>

## Programmieren mit Sage

Wenn man ein CAS (Computer-Algebra-System) nutzt, schreibt man zu Beginn einzelne Befehle in die Kommandozeile wie im obigen Beispiel<sup>15</sup>.

Wenn man eigene Funktionen entwickelt, sie ändert und aufruft, dann ist es viel einfacher, die Entwicklung in einem eigenen Editor vorzunehmen, den Code als Sage-Skriptdatei zu speichern und die Funktionen nicht-interaktiv auf der Kommandozeile auszuführen. Beide Arten, Code zu entwickeln, wurden in Kapitel 1.6 („Anhang: Beispiele mit Sage“), Kapitel 2.5 („Anhang: Beispiele mit Sage“), Kapitel 3.14 („Anhang: Beispiele mit Sage“) und in Kapitel 4.18 („Anhang: Beispiele mit Sage“) angewandt.

Um Sage-Code in einem eigenen Editor zu entwickeln und zu testen, gibt es zwei nützliche Befehle: `load()` und `attach()`<sup>16</sup>.

Angenommen Sie haben die folgende Funktions-Definition:

```
def function(var1):
    r"""
    DocText.
    """
    ...
    return (L)
```

die in der Datei `primroots.sage` gespeichert wurde.

Um diese Funktion zu laden (und syntaktisch gleich zu testen), wird der Befehl `load()` benutzt:

```
sage: load primroots.sage
```

Danach kann man auf der Kommandozeile alle Variablen und Funktionen nutzen, die im Sage-Skript definiert wurden<sup>17</sup>.

Normalerweise editiert man ein eigenes Sage-Skript wieder und möchte dann den Inhalt des geänderten Skripts wieder in Sage laden. Dafür kann man den Befehl `attach()` nutzen (man kann auch direkt nach dem `load()` das `attach()` aufrufen, und nicht erst, wenn man das Skript ändert; man kann `load()` sogar weglassen, da dies in `attach()` enthalten ist):

---

<sup>15</sup>Standardmäßig wird Sage-Code auch so präsentiert: Dabei beginnen die Zeilen mit „sage:“ und „...“.

```
sage: m = 11
sage: for a in xrange(1, m):
....:     print [power_mod(a, i, m) for i in xrange(1, m)]
....:
```

Auch dieses Skript benutzt normalerweise die obige Konvention, um Sage-Code zu präsentieren, solange der Code nicht aus einer Sage-Skriptdatei kommt. Wenn man den Sage-Code aus diesem Skript kopiert und per Paste auf der Sage-Kommandozeile wieder einfügt, sollte man „sage:“ und „...“ weglassen (obwohl das Kommandozeilen-Interface in den meisten Fällen mit diesen Präfixen korrekt umgehen kann).

<sup>16</sup>Vergleiche das Sage-Tutorial über Programmierung, Kapitel „Loading and Attaching Sage files“, <http://www.sagemath.org/doc/tutorial/programming.html#loading-and-attaching-sage-files>.

<sup>17</sup>Anmerkungen:

- Bitte keine Leerzeichen oder White Spaces im Dateinamen.
- Es empfiehlt sich, der Sage-Skriptdatei die Datei-Extension „.sage“ statt „.py“ zu geben. Hat ein Sage-Skript die Dateinamens-Endung „.sage“, dann wird beim Laden der Datei in Sage auch gleich die normale Sage-Umgebung mit geladen, um die Syntax zu prüfen. Genauso funktioniert es, wenn man ein Sage-Skript direkt von einer Bash-Shell aufruft mit `$ sage primroots.sage`.
- Beim Laden des obigen Sage-Skripts wird es von Sage zuerst geparkt, und dann in eine andere Datei namens „primroots.py“ kopiert. Sage ergänzt dann alle notwendigen Variablen in „primroots.py“ und alle Import-Statements. Somit wird das Sage-Skript genauso ausgeführt, als hätte man die Befehle einzeln auf der Kommandozeile eingetippt. Ein bedeutender Unterschied ist, dass alle Ausgaben ein `print` benötigen.

```
sage: attach primroots.sage
```

Nun kann man das Sage-Skript ändern und die geänderte Funktionsdefinition wird – solange man die Sage-Session nicht beendet – beim nächsten Enter in Sage geladen (und syntaktisch gleich geprüft). Diese Neuladen passiert vollkommen automatisch. Der Befehl `attach()` lässt Sage also permanent die genannte Datei auf Änderungen überwachen. Damit spart man sich das Kopieren und Pasten zwischen dem eigenen Texteditor und dem Sage-Kommandozeilen-Interface.

Hier ist ein Bild, das Sage-Code im Editor GVIM zeigt – mit aktiviertem Syntax-Highlighting (siehe Bild A.13).

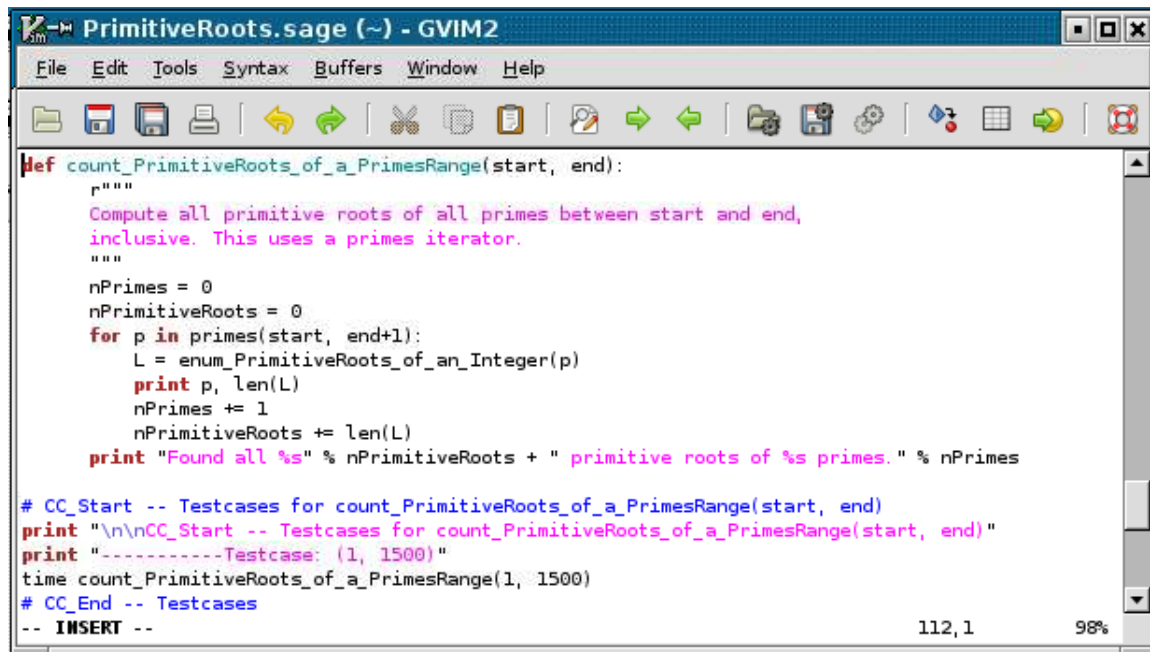


Abbildung A.13: Sage-Beispiel in einem Editor mit aktiviertem Code-Highlighting

Falls man die Ausgabe einer attachten Datei so angezeigt haben möchte, wie wenn man die Einzelbefehle direkt auf der Kommandozeile eingibt (also nicht nur das, was per `print` ausgegeben wird), kann man den Befehl `iload()` verwenden: Jede Zeile wird dann einzeln geladen. Um die nächste Zeile zu laden, muss man die **Enter**-Taste drücken. Das muss man so lange wiederholen, bis alle Zeilen des Sage-Skripts in die Sage-Session geladen sind.

```
sage: iload primroots.sage
```

Weitere Hinweise:

- Abfrage der Version Ihrer Sage-Umgebung mit: `version()`
- Um sich schnell die Sage-Programmbeispiele in diesem Skript anzusehen, können Sie
  - im Index nach **Sage** -> **Programmbeispiele** schauen, oder
  - sich im Anhang das „Verzeichnis der Sage-Programmbeispiele“ ansehen.
- Der Quellcode der Sage-Beispiele in diesem Skript wird in Form von Sage-Programmdateien mit dem CrypTool-Setup-Programm ausgeliefert. Nach der Installation von CrypTool 1.x finden Sie diese im Unterverzeichnis **sage** innerhalb des CrypTool-Verzeichnisses:
  - SAGE-Samples-in-Chap01.sage
  - SAGE-Samples-in-Chap02.sage
  - SAGE-Samples-in-Chap03.sage
  - SAGE-Samples-in-Chap04.sage

# GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to

the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:



- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## **8. TRANSLATION**

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## **9. TERMINATION**

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## **10. FUTURE REVISIONS OF THIS LICENSE**

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with ... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# Abbildungsverzeichnis

2.1	Namenskonventionen in den Sage-Programmbeispielen zu Verschlüsselungsverfahren . . . . .	34
2.2	Hill-Dialog in CrypTool mit den verfügbaren Operationen und Optionen . . . . .	48
3.1	Graph der Funktionen $x$ und $10^x$ . . . . .	85
3.2	Graph der Funktion $\ln x$ bis 100 und bis $10^{10}$ . . . . .	85
3.3	Die Funktionen $x$ (blau), $\ln x$ (rot) und $\frac{x}{\ln x}$ (grün) . . . . .	85
3.4	Anzahl der Primzahlen im Intervall 0 bis $10^x$ (blau) und im Interall $[10^{x-1}, 10^x]$ (rot) (für verschiedene Exponenten $x$ ). . . . .	86
4.1	Vergleich der publizierten Faktorisierungserfolge (blau) mit der prognostizierten Entwicklung (rot; Quelle Fox 2001) . . . . .	128
4.2	Die Anzahl der Primitivwurzeln für alle Primzahlen zwischen 1 und 100.000 . . .	167
4.3	Die kleinste Primitivwurzel von jeder Primzahl zwischen 1 und 100.000 . . . . .	168
4.4	Die größte Primitivwurzel von jeder Primzahl zwischen 1 und 100.000 . . . . .	168
7.1	Prognose für die Entwicklung der als sicher betrachteten Schlüssellängen für RSA und Elliptische Kurven . . . . .	201
7.2	Gegenüberstellung des Aufwands der Operationen Signieren und Verifizieren bei RSA und Elliptischen Kurven . . . . .	201
7.3	Beispiel einer Elliptischen Kurve über dem Körper der reellen Zahlen. . . . .	207
7.4	Verdoppelung eines Punktes . . . . .	210
7.5	Addition zweier verschiedener Punkte im Körper der reellen Zahlen . . . . .	210
A.1	Komplette Übersicht über den Menü-Baum von CrypTool 1.4.30 . . . . .	228
A.2	Jeder gemeinsame Teiler zweier Zahlen teilt auch alle ihre Linearkombinationen .	241
A.3	Euklids Algorithmus zur Bestimmung des ggT . . . . .	242
A.4	Verteilung der Primzahlen und ihrer Differenzen . . . . .	242
A.5	Primzahlen finden mit dem Primzahltest nach Fermat . . . . .	243
A.6	Umkehrbarkeit von Verschlüsselungsalgorithmen am Beispiel additiver Chiffren . .	243
A.7	Fermat-Faktorisierung m.H. der 3. Binomischen Formel . . . . .	244
A.8	Fermat-Faktorisierung: Quadrate erkennen . . . . .	244
A.9	Pollards Rho-Faktorisierung: Kreisfindungsalgorithmus nach Floyd . . . . .	245
A.10	Sage-Kommandozeilen-Interface . . . . .	248
A.11	Sage-Notebook-Interface . . . . .	248
A.12	Die Standard-Dokumentation von Sage . . . . .	250
A.13	Sage-Beispiel in einem Editor mit aktiviertem Code-Highlighting . . . . .	254

# Tabellenverzeichnis

2.1	Gartenzaun-Verschlüsselung . . . . .	15
2.2	8x8-Fleißner-Schablone . . . . .	16
2.3	Einfache Spaltentransposition . . . . .	17
2.4	Spaltentransposition nach General Luigi Sacco . . . . .	17
2.5	Nihilist-Transposition . . . . .	18
2.6	Cadenus . . . . .	19
2.7	Nihilist-Substitution . . . . .	21
2.8	Straddling Checkerboard mit Passwort ”‘Schluessel’” . . . . .	22
2.9	Variante des Straddling Checkers . . . . .	23
2.10	Baconian Cipher . . . . .	23
2.11	5x5-Playfair-Matrix . . . . .	25
2.12	Four Square Cipher . . . . .	26
2.13	Vigenère-Tableau . . . . .	27
2.14	Autokey-Variante . . . . .	27
2.15	Ragbaby . . . . .	28
2.16	Bifid . . . . .	29
2.17	Bazeries . . . . .	30
2.18	Digrafi . . . . .	30
2.19	Nicodemus . . . . .	31
3.1	Die 20+ größten Primzahlen und ihr jeweiliger Zahlentyp (Stand Juli 2009) . . .	56
3.2	Die größten vom GIMPS-Projekt gefundenen Primzahlen (Stand Juli 2009) . . .	60
3.3	Arithmetische Primzahlfolgen mit minimaler Distanz (Stand Jan. 2010) . . . .	74
3.4	Produkte der ersten Primzahlen $\leq k$ (genannt $k$ Primorial oder $k\#$ ) . . . . .	75
3.5	Wieviele Primzahlen gibt es innerhalb der ersten Zehnerintervalle? . . . . .	80
3.6	Wieviele Primzahlen gibt es innerhalb der ersten Dimensionsintervalle? . . . .	80
3.7	Liste selektierter $n$ -ter Primzahlen . . . . .	81
3.8	Wahrscheinlichkeiten und Größenordnungen aus Physik und Alltag . . . . .	82
3.9	Spezielle Werte des Zweier- und Zehnersystems . . . . .	83
4.1	Additionstabelle modulo 5 . . . . .	106
4.2	Multiplikationstabelle modulo 5 . . . . .	106
4.3	Multiplikationstabelle modulo 6 . . . . .	107
4.4	Multiplikationstabelle modulo 17 (für $a = 5$ und $a = 6$ ) . . . . .	108
4.5	Multiplikationstabelle modulo 13 (für $a = 5$ und $a = 6$ ) . . . . .	108
4.6	Multiplikationstabelle modulo 12 (für $a = 5$ und $a = 6$ ) . . . . .	109
4.7	Werte von $a^i \bmod 11, 1 \leq a, i < 11$ und zugehörige Ordnung von $a$ modulo $m$ . .	119
4.8	Werte von $a^i \bmod 45, 1 \leq a, i < 13$ . . . . .	119
4.9	Werte von $a^i \bmod 46, 1 \leq a, i < 23$ . . . . .	120
4.10	Die derzeitigen Faktorisierungsrekorde (Stand Jan. 2010) . . . . .	129

4.11	Großbuchstabenalphabet . . . . .	143
4.12	RSA-Geheimtext A . . . . .	147
4.13	RSA-Geheimtext B . . . . .	148
5.1	Eulersche Phi-Funktion . . . . .	183
5.2	Wertetabelle $L(N)$ . . . . .	184
5.3	Verfahren zur Berechnung des diskreten Logarithmus in $\mathbb{Z}_p^*$ . . . . .	185



# Verzeichnis der Krypto-Verfahren

5.1	Lösen von Knapsackproblemen mit superwachsenden Gewichten . . . . .	181
5.2	Merkle-Hellman (auf Knapsackproblemen basierend) . . . . .	182
5.3	RSA (auf dem Faktorisierungsproblem basierend) . . . . .	183
5.4	Rabin (auf dem Faktorisierungsproblem basierend) . . . . .	184
5.5	Diffie-Hellman-Schlüsselvereinbarung . . . . .	186
5.6	ElGamal (auf dem diskreten Logarithmusproblem basierend) . . . . .	187
5.7	Verallgemeinertes ElGamal (auf dem diskreten Logarithmusproblem basierend) .	189
6.1	DSA-Signatur . . . . .	196

# Verzeichnis der Sage-Programmbeispiele

1.1	Ver- und Entschlüsselung mit dem Mini-AES . . . . .	9
2.1	Einfache Transposition durch Shiften (die Schlüssel sind explizit gegeben) . . . .	35
2.2	Einfache Transposition durch Shiften (die Schlüssel werden mit "range" konstruiert) . . . . .	36
2.3	Einfache Spalten-Transposition mit zufällig erzeugtem (Permutations-) Schlüssel	37
2.4	Einfache Spalten-Transposition (mit Ausgabe der Größe des Schlüsselraumes) . .	38
2.5	Monoalphabetische Substitution mit zufällig erzeugtem Schlüssel . . . . .	39
2.6	Caesar (Substitution durch Shiften des Alphabets; Schlüssel explizit gegeben; Schritt-für-Schritt-Ansatz) . . . . .	40
2.7	Caesar (Substitution durch Shiften des Alphabets; Substitutions-Schlüssel wird berechnet) . . . . .	41
2.8	Verschiebe-Chiffre (über dem Großbuchstabenalphabet) . . . . .	42
2.9	Caesar-Verschlüsselung mit der Verschiebe-Chiffre . . . . .	42
2.10	Affine Chiffre mit dem Schlüssel (3, 13) . . . . .	43
2.11	Verschiebe-Chiffre (als Sonderfall der affinen Chiffre) . . . . .	43
2.12	Caesar-Chiffre (als Sonderfall der affinen Chiffre) . . . . .	44
2.13	Monoalphabetische Substitution über dem Binär-Alphabet . . . . .	45
2.14	Monoalphabetische Substitution über dem Hexadezimal-Alphabet (Dekodieren in Python) . . . . .	46
2.15	Vigenère-Verschlüsselung . . . . .	47
2.16	Hill-Verschlüsselung . . . . .	49
3.1	Spezielle Werte des Zweier- und Zehnersystems . . . . .	83
3.2	Erzeugen der Graphen zu den drei Funktionen $x$ , $\log(x)$ und $x/\log(x)$ . . . . .	87
3.3	Einige einfache Funktionen zu Primzahlen . . . . .	88
3.4	Testen der Primalität von Funktionswerten, erzeugt von einer quadratischen Funktion . . . . .	89
4.1	Multiplikationstabelle $a \times i \pmod m$ mit $m = 17$ , $a = 5$ and $a = 6$ . . . . .	155
4.2	Schnelles Berechnen hoher Potenzen mod $m = 103$ . . . . .	155
4.3	Tabelle mit allen Potenzen $a^i \pmod m$ für $m = 11$ , $a = 1, \dots, 10$ . . . . .	156
4.4	Tabelle mit allen Potenzen $a^i \pmod{45}$ für $a = 1, \dots, 12$ plus der Ordnung von $a$	157
4.5	Tabelle mit allen Potenzen $a^i \pmod{46}$ für $a = 1, \dots, 23$ plus die Ordnung von $a$ .	158
4.6	Berechnen einer Primitivwurzel jeweils für eine Primzahl . . . . .	159
4.7	Funktion "enum_PrimitiveRoots_of_an_Integer" zur Berechnung aller Primitiv- wurzeln für eine gegebene Zahl . . . . .	160
4.8	Tabelle mit allen Primitivwurzeln der vorgegebenen Primzahl 541 . . . . .	161
4.9	Funktion "count_PrimitiveRoots_of_an_IntegerRange" zur Berechnung aller Pri- mitivwurzeln für einen gegebenen Zahlenbereich . . . . .	162

4.10	Funktion “count_PrimitiveRoots_of_an_IntegerRange”: Testfälle und Testausgaben	163
4.11	Funktion “count_PrimitiveRoots_of_a_PrimesRange” zur Berechnung aller Prim- itivwurzeln für ein gegebenes Intervall von Primzahlen . . . . .	164
4.12	Code zur Erstellung einer Liste mit allen Primitivwurzeln für alle Primzahlen zwischen 1 und 100.000 . . . . .	165
4.13	Code zur Erzeugung der Grafiken zur Verteilung der Primitivwurzeln . . . . .	166
4.14	Faktorisierung einer Zahl . . . . .	169
4.15	RSA-Verschlüsselung durch modulare Exponentiation einer Zahl (als Nachricht) .	169
4.16	Wie viele RSA-Schlüssel gibt es, wenn man den Bereich für die Schlüsselgröße $n$ kennt? . . . . .	171
A.1	Einige kleine Beispiele in Sage aus verschiedenen Gebieten der Mathematik . . .	252

- Alle Beispiel wurden mit Sage v 4.2.1 (Release Date 2009-11-14) getestet.
- Der Quellcode der Sage-Beispiele in diesem Skript wird in Form von Sage-Programmdatei-  
en mit dem CrypTool-Setup-Programm ausgeliefert. Nach der Installation von CrypTool  
1.x finden Sie diese im Unterverzeichnis **sage** innerhalb des CrypTool-Verzeichnisses:
  - SAGE-Samples-in-Chap01.sage
  - SAGE-Samples-in-Chap02.sage
  - SAGE-Samples-in-Chap03.sage
  - SAGE-Samples-in-Chap04.sage

# Index

## A

Aaronson 2003, 91  
Abgeschlossenheit, 104, 112, 151  
ACA 2002, 50  
Addition, 105, 112  
ADFGVX, 28  
Adleman 1982, 190  
Adleman, Leonard, 6, 181, 182  
AES, 2, 6, 7  
    Mini-AES, 8  
Agrawal 2002, 173  
AKS, 78, 137  
Alice, 5, 139  
AMSCO, 16  
Angriff  
    Brute-Force, 2–4, 227  
    Chosen-Ciphertext, 184  
    Ciphertext-only, 146  
    Geburtstagsangriff, 193  
    Impersonalisierungsattacke, 195  
    Known-Plaintext, 146  
    Kollisionsangriff, 193  
    Man-in-the-Middle-Attack, 197  
    Pre-Image  
        1st, 192  
        2nd, 192  
Apted 2001, 235  
Arthur 196x, 232  
Assoziativgesetz, 103  
Atbash, 20  
Authentizität, 6, 197  
    Benutzer-, 191  
Autoren, 229

## B

Babystep-Giantstep, 185, 188  
Baconian Cipher, 23  
Balcazar 1988, 190  
Baldacci 1997, 233  
Bartholome 1996, 91, 173  
Bauer 1995, 50, 173  
Bauer 2000, 50, 173

BC, 141, 176  
Beale-Chiffre, 24  
Beaufort, 27  
Becker 1998, 234  
Beinaheprimzahl, 73  
Berne, Eric, 115  
Bernstein 2001, 173  
Beutelspacher 1996, 173  
Beweis  
    Existenzbeweis, 73  
    konstruktiv, 73  
Biryukov 2009, 10  
Blocklänge, 142, 143  
Blum 1999, 91  
Bob, 5, 139  
Bogk 2003, 175  
Bourseau 2002, 173  
Brands 2002, 173  
Brickell 1985, 190  
Brickell, Ernst, 181  
Brown 1998, 234  
Brown 2003, 236  
BSI, 122, 127, 131, 176  
BSI 2002, 173  
Buchchiffre, 24  
Buchmann 2004, 173, 246  
Buhler 1993, 174  
Bundschuh 1998, 91  
Burger 2006, 236

## C

C158, 130  
C307, 133  
Cadenus, 18  
Caesar, 20  
Caldwell, Chris, 93  
CAS, 83  
Cassels 1991, 219  
Catalan, Eugene, 69  
Certicom, 202, 209, 220  
Certification Authority (CA), 197  
Ché Guevara, 22

- Challenge, 5, 129
- Cipher-Challenge, 5, 129
- Cole, Frank Nelson, 57
- Colfer 2001, 235
- Coppersmith 2002, 10
- Courtois 2002, 10
- Crandall, Richard, 59
- Crandell 2001, 91
- Crichton 1988, 233
- Crowley 2000, 50
- CrypTool, ii, xi, xii, 2, 4–7, 15, 16, 20, 58, 60, 71, 101, 118, 122, 126, 130, 133, 139, 142–147, 149, 176, 178, 182, 185, 186, 191–193, 215, 227, 229, 234, 235, 241
- CrypTool 1.x, xi, 227, 255, 269
- CrypTool 2.0, 227
- Cunningham-Projekt, 63, 93, 130, 133, 177
- D**
- DA 1999, 50
- Dedekind, Julius, 97
- DES, 2, 4, 7
  - SDES, 7
- Diffie, Whitfield, 6, 139, 186
- Diffie-Hellman, 96, 139, 186, 187, 212
- Diskreter Logarithmus, 111, 141, 185
- Distributivgesetz, 103
- Division modulo  $n$ , 103, 105
- Divisor, 101
- DL-Problem, 111, 141, 185
- Domain-Parameter, 212
- Doppelwürfel, 16
- Doyle 1905, 232
- Doyle, Sir Arthur Conan, 232
- DSA, 6, 213, 215
- DSA-Signatur, 6, 195
- E**
- ECDLP, 211, 212
- Eckert 2003, 122, 143, 174
- ECMNET, 213
- EFF, 60
- Einwegfunktion, 112, 138, 178
  - mit Falltür, 179
- ElGamal
  - Public-Key, 187
- ElGamal, Tahir, 6
- Elliptische Kurven, 200
  - ECC-Notebook, 216
- Elsner 1999, 234
- Elsner 2001, 235
- Eratosthenes
  - Sieb, 60, 71
- Erdős, Paul, 72
- Ertel 2001, 174
- Eschbach 2009, 237
- Euklid, 54
- Euklid's Widerspruchsbeweis, 55
- Euklidischer Algorithmus, 214
  - erweiterter, 107, 116, 149, 214
- Euklidzahlen, 66
- Euler, Leonhard, 115, 116
- Eulersche Phi-Funktion, 107, 111, 115, 182
- Exponentialfunktion
  - Berechnung, 187
  - diskrete, 185
- F**
- Faktor, 101
- Faktorisierung, 58, 203, 233
  - Faktorisierungsalgorithmen, 213
  - Faktorisierungsproblem, 117, 123, 125, 135, 146, 182, 184
  - Faktorisierungsrekorde, 58, 78, 129, 176, 213
  - Prognose, 127
- Ferguson 2001, 10
- Fermat
  - Fermat-Primzahl, 64
  - Fermatzahl, 61, 64
    - verallgemeinerte, 55, 56, 65
  - kleiner Satz, 61, 107, 116
  - letzter Satz, 97, 175, 202
- Fermat, Pierre, 61, 116, 202
- Fibonacci, 96, 176
- Filme, 79, 121, 231
- FIPS180-3, 198
- FIPS186, 198
- FIPS186-2, 198
- Fixpunkt, 115, 117
- Fleißner-Schablone, 16
- Flessner 2004, 240
- FlexiProvider, 223
- Fox 2002, 173
- Fox, Dirk, 127
- G**
- Gödel, Kurt, 76
- Gallot, Yves, 63, 65
- Gartenzaun-Verschlüsselung, 15

Gauss, Carl Friedrich, 64, 70, 95, 96, 100, 121  
 Gaussklammer, 72, 149  
 General Number Field Sieve (GNFS), 125, 126, 129–132, 135, 137, 186  
 Gesetz von Moore, 127, 200  
 ggT, 96, 107, 111, 149  
 GIMPS, 59, 93  
 Gitterbasenreduktion, 126  
 Goebel 2003, 50  
 Goldbach, Christian, 75  
 Goldbach-Projekt, 93  
 Google  
   Mitarbeiterwerbung, 78  
 Graham 1989, 91  
 Graham 1994, 96, 174  
 Grid-Computing, 127  
 Großbuchstabenalphabet, 143, 148  
 Gruppen, 112, 187, 203  
   Ordnung, 204  
   zyklische, 204

## H

Halbprimzahl, 73, 170  
 Harder 2003, 239  
 Hardy, Godfrey Harold, 72, 73  
 Hashfunktion, 192  
 Hashwert, 192  
 Hellman, Martin, 6, 139, 181, 185, 186  
 Hesselink 2001, 190  
 Hill 1929, 50  
 Hill 1931, 50  
 Hill 2009, 237  
 Hoffman 2006, 10  
 Howard 2001, 235  
 Hybridverfahren, 6

## I

IDEA, 2, 7  
 Identität, 104  
 IETF, 7  
 Impersonalisierungsattacke, 195  
 Inverse  
   additive, 104, 106, 108  
   multiplikative, 104, 106, 108  
 invertierbar, 118  
 Isau 1997, 235  
 ISO/IEC 9594-8, 199  
 ITU-T, 199  
 IVBB, 217

## J

JCrypTool 1.0, 227

## K

Körper, 203  
   Charakteristik, 205  
   endliche, 205  
 Kaskaden, 3, 28  
 Kipling 1901, 231  
 Kipling, Rudyard, 231  
 Kippenhahn 1997, 174  
 Kippenhahn 1999, 174  
 Kippenhahn 2002, 239  
 Klee 1997, 91  
 Kleinjung 2010, 174  
 Knapsack, 180  
   Merkle-Hellman, 181  
 Knott, Ron, 96, 176  
 Knuth 1981, 91  
 Knuth 1998, 174  
 Koblitz 1984, 219  
 Koblitz 1998, 219  
 Koblitz, Neal, 203  
 Kollision, 192, 193  
 Kommutativgesetz, 103  
 Komplexität, 125, 179, 188, 203  
   Komplexitätsklasse, 125  
   subexponentielle, 125  
 Kongruenz, 101, 102  
 Kronecker, Leopold, 97  
 Krypto-Wettbewerb, 5, 129  
 Kryptoanalyse, 2, 7, 143, 146  
 Kryptographie  
   moderne, 52, 138, 178  
   Post Quantum, 223  
   Public-Key, 52, 121, 180

## L

Lagarias 1983, 190  
 Lagarias, Jeff, 181  
 Landkarten-Chiffre, 21  
 Larsson 2007, 237  
 Laufzeit  
   effizient, 179  
   nicht polynomial NP, 180  
   polynomial, 179  
 Legendre, Adrien-Marie, 70, 121  
 Lem, Stanislaw, 192  
 Lenstra 1987, 213, 219  
 Lenstra 1993, 174

Lenstra 2002, 174  
 Lenstra/Verheul, 221  
 Lenstra/Verheul 1999, 174, 219  
 Lernprogramm ZT, 60, 71, 101, 118, 126, 149, 178, 185, 241  
 Lichtenberg, Georg Christoph, 178  
 LiDIA, 141, 176  
 Literatur, 121, 231  
 Logarithmieren, 111  
 Logarithmusproblem, 212  
     diskret, 111, 140, 141, 185, 188, 195  
     Rekord, 185  
 Long-Integer, 110  
 Lorenz 1993, 91  
 Lucas, Edouard, 57, 60  
 Lucks 2002, 10  
 Lucks 2003, 175

## M

M1039, 133  
 Müller-Michaelis 2002, 239  
 Mansoori, Bizaki 2007, 10  
 Massierer, Maike, 216  
 Mathematica, 141, 176  
 McBain 2004, 236  
 McDonald 2009, 198  
 Mehrfachverschlüsselung, 3, 28  
 Menezes 1993, 219  
 Menezes 2001, 174  
 Merkle 1978, 190  
 Merkle, Ralph, 181  
 Mersenne  
     Mersenne-Primzahl, 57, 58, 63, 78, 93  
     M-37, 58  
     M-38, 58  
     M-39, 59, 63  
     Mersennezahl, 57  
         verallgemeinerte, 55, 63, 64  
     Satz, 57  
 Mersenne, Marin, 57, 61  
 Miller Gary L., 62  
 Miller, Victor, 203  
 Modulus, 101  
 Moore, Gordon E., 127, 200  
 Moses xxxx, 239  
 Multiplikation, 105, 113  
 Münchenbach, Carsten, 176  
 Musa, Schaefer, Wedig 2003, 10

## N

Nachrichtenintegrität, 191  
 Natali 1997, 233  
 Nguyen 2009, 11, 51  
 Nguyen, Minh Van , 142  
 Nichols 1996, 11, 50  
 Nihilist-Substitution, 20  
 Nihilist-Transposition, 18  
 NIST, 193, 195  
 Noll, Landon Curt, 58  
 Nomenklatur, 21  
 NSA, 2, 7

## O

One-Time-Pad, 1  
 Open Source, 122  
 Oppliger 2005, 91  
 Ordnung  
     maximale, 118  
     multiplikative, 118

## P

P(n), 70  
 Padberg 1996, 91  
 Palladium, 137  
 Papier- und Bleistiftverfahren, 14, 235  
 Para 1988, 239  
 Pari-GP, 141, 176, 177  
 Patent, 122, 217  
 Performance, 192, 200  
 Permutation, 15, 108, 120, 181  
 Pfleeger 1997, 174  
 Phan 2002, 11  
 Phan 2003, 11  
 PI(x), 70  
 Pieper 1983, 92  
 PKCS#1, 195, 198  
 PKCS#5, 192  
 PKI, 195  
 Playfair, 24  
 Poe 1843, 231  
 Poe, Edgar Allan, 14, 231  
 Pohlig, S. C., 185  
 Pollard, John M., 213  
 Polynom, 68, 78, 125, 137, 179–181, 206  
 Pomerance 1984, 174  
 Pomerance 2001, 91  
 Potenz, 110  
 Potenzieren, 109  
 Pre-Image-Attack

- 1st, 192
- 2nd, 192
- Primfaktor, 53, 100
  - Zerlegung, 100, 111, 115, 182
- Primitivwurzel, 107, 108, 118–120, 141, 159
- Primzahl, 52, 99
  - Anzahl, 121
  - Beinaheprimzahl, 73
  - Dichte, 69
  - Fermat, 64
  - Formel, 63
  - gigantische, 58
  - Halbprimzahl, 73, 170
  - Mersenne, 58, 63, 78
  - Pseudoprimzahl, 61, 65
  - relative, 66, 183
  - semiprime, 129, 170
  - starke Pseudoprimzahlen, 62, 65
  - titanische, 58
- Primzahlfolge
  - arithmetische, 72
- Primzahlrekorde, 55
- Primzahlsatz, 70, 121
- Primzahltest, 58, 60, 78, 134, 137, 203
- Produktalgorithmus, 3
- Pythagoras
  - Satz von, 97

## Q

- Quadratic Sieve-Algorithmus (QS), 126
- Quantencomputer, 222–224
- Quantenkryptographie, 224

## R

- Rabin
  - Public-Key-Verfahren, 184
- Rabin, Michael O., 62, 184
- RC5, 4
- Reduzierbarkeit, 103
- relativ prim, 66, 107
- restgleich, 102
- Restklasse, 101
- Restmenge
  - reduzierte, 113
  - vollständige, 113
- Richstein 1999, 75, 92
- Riemann, Bernhard, 76
- RIPEMD-160, 193
- Rivest, Ronald, 6, 182
- Robinson 1992, 233

- Robshaw 2002, 11
- Rowling, Joanne, 99, 138
- RSA, 6, 52, 96, 116, 117, 121, 122, 142, 182, 221
  - Cipher-Challenge, 146
  - Modulus, 212
  - Signatur, 124, 194
- RSA 1978, 190
- RSA Laboratories, 198, 202
- RSA Security 2002, 175
- RSA-155, 130
- RSA-160, 131
- RSA-200, 132
- RSA-768, 132

## S

- Sage, ii, 14, 34, 67, 83, 84, 88, 89, 108, 141, 142, 155, 177, 215, 220, 247
  - Programmbeispiele, 8, 34, 84, 88, 155, 215, 247
- Savard 1999, 51
- Sayers 1932, 232
- Schaefer 1996, 11
- Scheid 1994, 92
- Scheid 2003, 246
- Schlüssel
  - öffentlich, 5, 179
  - geheim, 5
  - privat, 179
- Schlüsselaustausch
  - Diffie-Hellman, 139, 186
- Schlüsselmanagement, 6
- Schmeh 2004, 51
- Schmeh 2007, 51
- Schmeh 2009, 11
- SchneiderM 2004, 175
- Schneier 1996, 11, 92, 175, 198
- Schnorr, C.P., 6
- Schrödel, Tobias, 238, 240
- Schröder 2008, 237
- Schroeder 1999, 92
- Schwenk 1996, 92
- Schwenk 2002, 175
- SECUDE IT Security GmbH, 6
- Sedgewick 1990, 122, 175
- Seed 1990, 233
- Seneca, 105
- Session Key, 6
- Seventeen or Bust SoB, 56
- SHA-1, 193, 196



Shamir 1982, 190  
 Shamir 2003, 175  
 Shamir 2003a, 175  
 Shamir, Adi, 6, 181, 182  
 Short-Integer, 110  
 Shoup 2005, 92  
 Sicherheit  
     langfristige, 221  
     Voraussage, 222  
 Signatur  
     digitale, 6, 52, 124, 191, 194, 195  
     DSA, 6, 195  
     Merkle, 224  
     RSA, 124, 194  
 Signaturverfahren, 191  
 Silver, 185  
 Silverman 1986, 219  
 Silverman 1992, 219  
 Silverman 2000, 175  
 Silverman/Tate 1992, 219  
 Simmel 1970, 232  
 Singh 2001, 51  
 Skytale, 15  
 Smartcard, 200  
 Solitaire, 32  
 Special Number Field Sieve (SNFS), 130, 133  
 Square and multiply, 111, 146  
 Stallings 2006, 11  
 Stamp 2007, 12  
 Standardisierung, 202  
 Steganographie, 21  
 Stephenson 1999, 234  
 Stinson 1995, 146, 175, 190  
 Straddling Checkerboard, 22  
 Struktur, 104, 112, 115, 118  
 Substitution, 20  
     homophon, 23  
     monoalphabetisch, 20  
     polyalphabetisch, 26  
     polygraphisch, 24  
 Superposition, 27  
 Swenson 2008, 12  
  
**T**  
 Talke-Baisch 2003, 240  
 Teilbarkeit, 101  
 Teiler, 101  
 ThinkQuest 1999, 51  
 Tietze 1973, 92  
 Transitivität, 104  
  
 Transposition, 15  
 Triple-DES, 4  
 TWIRL-Device, 136, 175  
  
**U**  
 Umkehrbarkeit, 114  
  
**V**  
 Verne 1885, 231  
 Verne, Jules, 231  
 Verschlüsselung, 1  
     asymmetrisch, 5, 121  
     codebasiert, 223  
     ElGamal-Public-Key, 187  
     Gitterprobleme, 223  
         NTRU, 223  
     hybrid, 6  
     McEliece, 223  
     Mehrfachverschlüsselung, 3, 28  
     Merkle-Hellman, 181  
     Produktalgorithmus, 3  
     Public-Key, 179  
     Superposition, 27  
     symmetrisch, 2, 14  
 Vidal 2006, 236  
 Vigenère, 26  
  
**W**  
 Wang 2005, 198  
 Weierstrass, Karl, 207, 208  
 Weis 2003, 175  
 Welschenbach 2001, 175  
 Wertebereich, 108, 120  
 Widerspruchsbeweis, 54, 55, 57  
 Wiles, Andrew, 97, 175, 202  
 Wobst 2002, 12  
 Wobst 2005, 198  
 Wolfenstetter 1998, 175  
 Woltman, George, 59  
 Wurzel, 111  
  
**X**  
 X.509, 197–199  
  
**Y**  
 Yan 2000, 146, 175  
 Yates, Samuel, 58  
  
**Z**  
 $\mathbb{Z}_n$ , 112  
 $\mathbb{Z}_n^*$ , 113

- Zübert 2005, 240
- Zahlen, 52
  - Carmichaelzahl, 62, 65
  - Catalanzahl, 69
  - Fermatzahl, 61, 64
  - Halbprimzahl, 73
  - Mersennezahl, 57
  - natürliche, 96
  - Primzahl, 52, 53
  - Proth-Zahl, 63
  - Pseudoprimzahl, 61, 65
  - semiprime, 73, 129, 170
  - Sierpinski, 56, 63
  - starke Pseudoprimzahl, 62, 65
  - teilerfremde (co-prime), 103, 107, 108, 113, 115–120, 122, 123, 142, 144, 150, 157, 180, 182, 183
  - zusammengesetzte, 53, 99
- Zahlentheorie
  - Einführung, 96
  - elementare, 95, 99
  - Hauptsatz, 53, 100
  - moderne, 97
- Zemeckis 1997, 79
- Zertifizierung
  - Public-Key, 195
- ZT, Lernprogramm Zahlentheorie, 60, 71, 101, 118, 126, 149, 178, 185, 241
- Zufall, 6, 196