

The top section of the cover features a dark blue background with glowing binary code (0s and 1s) and several overlapping circles in shades of blue and orange, creating a digital or network-like aesthetic.

Duncan Buell

Grundlagen der Kryptographie

Einführung in die mathematischen und
algorithmischen Grundlagen



Springer Vieweg

Grundlagen der Kryptographie

Duncan Buell

Grundlagen der Kryptographie

Einführung in die mathematischen und
algorithmischen Grundlagen



Springer Vieweg

Duncan Buell
Department of Computer Science and
Engineering
University of South Carolina
Columbia, SC, USA

ISBN 978-3-031-50431-0 ISBN 978-3-031-50432-7 (eBook)

<https://doi.org/10.1007/978-3-031-50432-7>

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Dieses Buch ist eine Übersetzung des Originals in Englisch „Fundamentals of Cryptography“ von Duncan Buell, publiziert durch Springer Nature Switzerland AG im Jahr 2021. Die Übersetzung erfolgte mit Hilfe von künstlicher Intelligenz (maschinelle Übersetzung). Eine anschließende Überarbeitung im Satzbetrieb erfolgte vor allem in inhaltlicher Hinsicht, so dass sich das Buch stilistisch anders lesen wird als eine herkömmliche Übersetzung. Springer Nature arbeitet kontinuierlich an der Weiterentwicklung von Werkzeugen für die Produktion von Büchern und an den damit verbundenen Technologien zur Unterstützung der Autoren.

Übersetzung der englischen Ausgabe: „Fundamentals of Cryptography“ von Duncan Buell, © Springer Nature Switzerland AG 2021. Veröffentlicht durch Springer International Publishing. Alle Rechte vorbehalten.

© Der/die Herausgeber bzw. der/die Autor(en), exklusiv lizenziert an Springer Nature Switzerland AG 2024

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von allgemein beschreibenden Bezeichnungen, Marken, Unternehmensnamen etc. in diesem Werk bedeutet nicht, dass diese frei durch jedermann benutzt werden dürfen. Die Berechtigung zur Benutzung unterliegt, auch ohne gesonderten Hinweis hierzu, den Regeln des Markenrechts. Die Rechte des jeweiligen Zeicheninhabers sind zu beachten.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen. Der Verlag bleibt im Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutionsadressen neutral.

Planung/Lektorat: Wayne Wheeler

Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Nature Switzerland AG und ist ein Teil von Springer Nature.

Die Anschrift der Gesellschaft ist: Gewerbestrasse 11, 6330 Cham, Switzerland

Das Papier dieses Produkts ist recyclebar.

Vorwort

Ich habe in einer Informatikabteilung unterrichtet. Unser Kryptographiekurs, den ich gerade zum vierten Mal abgeschlossen habe, ist für Studierende im höheren Semester (Junioren und Senioren) und auch für Graduiertenkredit. Dies ist, denke ich, ähnlich dem, was in einer großen Anzahl von Universitäten in den Vereinigten Staaten verfügbar ist. Unser Kurs ist auch mit der Mathematikabteilung verknüpft, und wir haben normalerweise eine Kombination von Mathematik- und Informatikstudenten, die den Kurs belegen.

Die Informatikstudenten wissen, wie man programmiert, aber sie sind oft nicht so gut in Mathematik. Im Gegensatz dazu können die Mathematikstudenten normalerweise die Mathematik bewältigen, haben aber nicht die gleiche Erfahrung in der Programmierung. Ich habe normalerweise Programmieraufgaben in Python, C++ oder Java akzeptiert. Ich habe darauf hingewiesen, dass Python standardmäßig eine mehrpräzise Ganzzahlarithmetik bietet und habe die Studenten, die C++ verwenden, auf das `gmp` Paket und die Studenten, die Java verwenden, auf das `BigNum` Paket hingewiesen. Ich habe die Studenten auch ermutigt, in Gruppen an den Programmieraufgaben zu arbeiten, wodurch die Gruppen sowohl das mathematische Verständnis als auch die Fähigkeit, die vorgestellten Algorithmen und Heuristiken zu programmieren, kombinieren können.

Ich habe ein minimales Hintergrundwissen in der asymptotischen Analyse von Algorithmen vorausgesetzt. Ich denke, ich habe das große $\mathcal{O}(*)$ auf ein Minimum reduziert, und ich denke, dass praktisch die gesamte benötigte diskrete Mathematik (modulare Arithmetik, Gruppen, Ringe und dergleichen) tatsächlich in diesem Text erscheint.

Ich denke, die meisten Schüler in den Vereinigten Staaten sehen mindestens die Grundlagen der Matrixreduktion in der High School; Ich habe versucht, die lineare Algebra auf das zu beschränken, was die meisten Schüler tatsächlich in der High School sehen könnten; und ich habe keine lineare Algebra über den mechanischen Prozess der Matrixreduktion hinaus verwendet.

Obwohl es einen gewissen Hintergrund in Gruppen und Ringen gibt, habe ich versucht, das so einfach wie möglich zu gestalten. Ein Vorteil dieses Materials ist, dass es nicht reine Theorie ist; man kann tatsächlich konkrete Beispiele für die Theoreme sehen

und, ich würde hoffen, verstehen, dass die Theorie hauptsächlich nur eine Art ist, das Material zu diskutieren und nicht das Ende an sich.

Es ist tatsächlich genau deshalb, weil dieses Material in beiden Abteilungen erscheinen sollte und tut, dass ich mich entschieden habe, dieses Buch zu schreiben. Es gibt mehrere gute Krypto-Bücher, die für einen rein mathematischen Kurs geeignet sind. Es gibt mindestens eines, das für einen niedrigeren Kurs geeignet ist, der sich an ein allgemeineres Publikum richtet.

Kryptographie, wie sie in diesem Jahrhundert praktiziert wird, ist stark mathematisch, hat aber auch Wurzeln in dem, was rechnerisch machbar ist. Was ich hoffe, dass dieser Text tut, was ich denke, dass andere Texte nicht tun, ist das Ausbalancieren der Theoreme der Mathematik gegen die Machbarkeit der Berechnung. Dies ist die Essenz meiner Einbeziehung des Materials von Kap. 8 in den Text. Mein Hintergrund ist Mathematik. Meine Erfahrung (mit Ausnahme eines ersten Jahres in einer Mathematikabteilung nach meiner Promotion) liegt in Informatikabteilungen, mit 15 Jahren in einem Forschungslabor, das Hochleistungsrechenforschung zur Unterstützung der National Security Agency durchführt. Meine „Sicht“ auf Kryptographie, einschließlich meiner Zeit bei der NSA, ist, dass es etwas ist, das man tatsächlich „macht“, kein mathematisches Spiel, über das man Theoreme beweist. Es gibt tiefe Mathematik; es gibt einige Theoreme, die bewiesen werden müssen; und es besteht die Notwendigkeit, die brillante Arbeit derjenigen anzuerkennen, die sich auf die Theorie konzentrieren. Aber auf der Ebene eines Bachelor-Kurses denke ich, dass der Schwerpunkt zunächst auf dem Kennen und Verstehen der Algorithmen und deren Implementierung liegen sollte, und auch darauf, sich bewusst zu sein, dass die Algorithmen sorgfältig implementiert werden müssen, um die „einfachen“ Wege zu vermeiden, die Kryptographie zu brechen.

Übungen

Die meisten Übungen in den späteren Kapiteln sind Programmieraufgaben. Dies ist hauptsächlich auf die Schwierigkeit zurückzuführen, realistische Probleme zu stellen, die vollständig von Hand erledigt werden können. Was die Rechenressourcen betrifft: Viele der Übungen, und viele der notwendigen Übungen, erfordern eine gewisse Berechnung. Es gibt viele Werkzeuge, die verwendet werden können. Das (alte, aber sicherlich machbare) Unix-Werkzeug `bc` wird die für viele der Berechnungen notwendige Mehrfachpräzisionsarithmetik durchführen und kann mit etwas Ähnlichem skriptgesteuert werden

```
bc < myscript.txt
```

um von der Standard-Eingabe umzuleiten. Einige Studenten werden Matlab® oder Mathematica oder Maple bevorzugen. Andere finden vielleicht SageMath ein besseres Werkzeug. Der Schlüssel für einige der einfacheren Probleme besteht darin, dass die Studenten die Software als einen ausgefeilten Taschenrechner verwenden, aber die Algorithmen selbst durchführen, anstatt nur Funktionen aufzurufen, die von jemand

anderem erstellt wurden. Der Schlüssel für die echten Programmierprobleme besteht darin, dass die Studenten die Details der Algorithmen lernen, zumindest für kleine Beispiele, die keine multiprätzisen Rechenpakete erfordern würden. Nach dem klassischen Rat, es „richtig zu machen, bevor man es besser macht“, wenn die Studenten Einzelprätzisionsbeispiele codieren können, die funktionieren, dann wird der größte Teil der Arbeit, um Mehrfachprätzisionsbeispiele zu codieren, nur der Übergang von Einzel- zu Mehrfachprätzision sein, und die Nachverfolgung des Einzelprätzisionsbeispiels kann als Grundwahrheit dafür verwendet werden, ob die Mehrfachprätzisionsversion Fehler hat.

Danksagungen

Es wäre unangemessen, wenn ich nicht meinem neuesten Kryptographiekurs danken würde, für den die erste Version dieses Buches als Lehrbuch diente, und den vorherigen Klassen, für die frühere Versionen geschrieben wurden. Ich habe großartiges Feedback von der neuesten Klasse erhalten, einige Korrekturen und einige Vorschläge, was geändert werden sollte und was geändert werden musste, um lesbarer zu werden. Jeder, der unterrichtet, wird sich über einige der Schüler beschweren, aber ich hatte das Glück, mit einer Reihe von Superstars zu unterrichten und zu arbeiten; Ich bin dankbar, solche Schüler gehabt zu haben.

Columbia, SC, USA
Februar 2021

Duncan Buell

Inhaltsverzeichnis

- 1 Einführung** 1
 - 1.1 Geschichte 1
 - 1.2 Einführung 4
 - 1.3 Warum wird Kryptographie verwendet? 5
 - 1.4 Verschlüsselungsmodi 7
 - 1.5 Angriffsmodi 8
 - 1.6 Wie viele Sekunden hat ein Jahr? 8
 - 1.7 Kerckhoffs' Prinzip 10
 - 1.8 Übungen 11
 - Literatur 11
- 2 Einfache Chiffren** 13
 - 2.1 Substitutionschiffren 14
 - 2.1.1 Caesar-Verschlüsselungen 14
 - 2.1.2 Zufällige Substitutionen 15
 - 2.1.3 Vigenère als Beispiel für polyalphabetische Substitutionen 15
 - 2.2 Spracheigenschaften und -muster 17
 - 2.2.1 Buchstabenhäufigkeit 17
 - 2.2.2 Wortgrenzen 18
 - 2.2.3 Cribbing 19
 - 2.2.4 Entropie 19
 - 2.3 Transpositionschiffren 23
 - 2.3.1 Säulentranspositionen 23
 - 2.3.2 Doppelte Transposition 24
 - 2.4 Playfair 24
 - 2.5 ADFGX 25
 - 2.6 Kryptoanalyse 25
 - 2.6.1 Das Knacken eines Substitutions-Chiffres 25
 - 2.6.2 Das Knacken eines Transpositionsschlüssels 26

2.7	Das Vernam Einmal-Pad	27
2.8	Übungen	28
2.8.1	Chiffretext für Substitutionschiffre-Probleme (3) und (4) . . .	29
	Literatur	29
3	Teilbarkeit, Kongruenzen und modulare Arithmetik	31
3.1	Teilbarkeit	31
3.2	Der euklidische Algorithmus	34
3.2.1	Der naive euklidische Algorithmus	34
3.2.2	Der erweiterte euklidische Algorithmus	35
3.2.3	Der Binäre Euklidische Algorithmus	36
3.2.4	Der Dreimal-Subtrahieren-Euklidische Algorithmus	38
3.2.5	GGT von großen Ganzzahlen	39
3.3	Primzahlen	40
3.4	Kongruenzen	41
3.5	Die Eulersche Totient	49
3.6	Fermats kleiner Satz	50
3.7	Exponentiation	50
3.8	Matrixreduktion	52
3.9	Übungen	53
	Literatur	54
4	Gruppen, Ringe, Felder	57
4.1	Gruppen	57
4.2	Ringe	62
4.3	Felder	62
4.4	Beispiele und Erweiterungen	63
4.4.1	Arithmetik modulo Primzahlen	63
4.4.2	Arithmetik Modulo zusammengesetzter Zahlen	66
4.4.3	Endliche Körper der Charakteristik 2	69
4.5	Übungen	69
	Literatur	70
5	Quadratwurzeln und quadratische Symbole	71
5.1	Quadratwurzeln	71
5.1.1	Beispiele	73
5.2	Charaktere auf Gruppen	75
5.3	Legendre-Symbole	76
5.4	Quadratische Reziprozität	77
5.5	Jacobi-Symbole	77
5.6	Erweitertes Gesetz der quadratischen Reziprozität	77
5.7	Übungen	79
	Literatur	80

6	Endliche Felder der Charakteristik 2	81
6.1	Polynome mit Koeffizienten mod 2	81
6.1.1	Ein Beispiel	81
6.2	Lineare Rückkopplungsschieberegister	83
6.3	Die allgemeine Theorie	88
6.4	Normale Basen	90
6.5	Übungen	94
	Literatur	94
7	Elliptische Kurven	97
7.1	Grundlagen	97
7.1.1	Gerade Linien und Schnittpunkte	98
7.1.2	Tangentenlinien	100
7.1.3	Formeln	100
7.1.4	Die Mordell-Weil Gruppe	101
7.2	Beobachtung	105
7.3	Projektive Koordinaten und Jacobische Koordinaten	105
7.4	Ein Beispiel für eine Kurve mit vielen Punkten	106
7.5	Kurven Modulo einer Primzahl p	107
7.6	Hasse's Theorem	108
7.7	Übungen	109
	Literatur	110
8	Mathematik, Informatik und Arithmetik	111
8.1	Mersenne Primzahlen	111
8.1.1	Einführung	112
8.1.2	Theorie	112
8.1.3	Implementierung	115
8.1.4	Zusammenfassung: Machbarkeit	117
8.1.5	Fermat-Zahlen	117
8.1.6	Der arithmetische Trick ist wichtig	118
8.2	Multipräzise Arithmetik und die schnelle Fourier-Transformation	118
8.2.1	Multipräzise Arithmetik	118
8.2.2	Hintergrund der FFT	119
8.2.3	Polynommultiplikation	119
8.2.4	Komplexe Zahlen wie benötigt für Fourier-Transformationen	120
8.2.5	Die Fourier-Transformation	121
8.2.6	Die Cooley-Tukey Schnelle Fourier-Transformation	122
8.2.7	Ein Beispiel	123
8.2.8	Der FFT-Schmetterling	129

8.3	Montgomery-Multiplikation	132
8.3.1	Der Rechenvorteil	134
8.4	Allgemeine Arithmetik	135
8.5	Übungen	135
	Literatur	135
9	Moderne symmetrische Chiffren – DES und AES	137
9.1	Geschichte	137
9.1.1	Kritik und Kontroverse	138
9.2	Der Advanced Encryption Standard	139
9.3	Der AES-Algorithmus	141
9.3.1	Polynomiale Grundlagen: Das Galois-Feld $GF(2^8)$	142
9.3.2	Byte-Organisation	143
9.4	Die Struktur von AES	144
9.4.1	Die äußere Struktur der Runden	144
9.4.2	Allgemeine Code-Details	145
9.4.3	Schlüsselerweiterung	145
9.4.4	SubBytes	148
9.4.5	ShiftRows	151
9.4.6	MixColumns	153
9.4.7	AddRoundKey	156
9.5	Implementierungsprobleme	156
9.5.1	Softwareimplementierungen	157
9.5.2	Hardware-Implementierungen	159
9.6	Sicherheit	161
9.7	Übungen	162
	Literatur	163
10	Asymmetrische Chiffren – RSA und andere	165
10.1	Geschichte	165
10.2	RSA Public-Key Verschlüsselung	166
10.2.1	Der grundlegende RSA-Algorithmus	167
10.3	Implementierung	168
10.3.1	Ein Beispiel	168
10.4	Wie schwer ist es, RSA zu knacken?	170
10.5	Andere Gruppen	171
10.6	Übungen	172
	Literatur	172
11	Wie man eine Zahl faktorisiert	173
11.1	Pollard Rho	174
11.2	Pollard $p-1$	176
11.2.1	Die allgemeine Metaphysik von $p-1$	177
11.2.2	Schritt Zwei von $p-1$	177

11.3	CFRAC	178
11.3.1	Fortgesetzte Brüche	179
11.3.2	Der CFRAC-Algorithmus	182
11.3.3	Beispiel	184
11.3.4	Berechnung	186
11.4	Faktorisierung mit elliptischen Kurven	187
11.5	Übungen	188
	Literatur	189
12	Wie man effektiver faktorisiert	191
12.1	Mängel von CFRAC	191
12.2	Das Quadratische Sieb	192
12.2.1	Der Algorithmus	192
12.2.2	Die entscheidenden Gründe für Erfolg und Verbesserung gegenüber CFRAC	193
12.3	Noch einmal in die Bresche	193
12.4	Das Mehrfachpolynom Quadratische Sieb	194
12.4.1	Noch ein weiterer Vorteil	195
12.5	Die Zahlkörpersieb	196
12.6	Übungen	196
	Literatur	197
13	Zyklen, Zufälligkeit, diskrete Logarithmen und Schlüsselaustausch	199
13.1	Einführung	199
13.2	Das diskrete Logarithmusproblem	200
13.3	Schwierige Diskrete Logarithmusprobleme	201
13.4	Zyklen	202
13.5	Cocks-Ellis-Williamson/Diffie-Hellman Schlüsselaustausch	203
13.5.1	Der Schlüsselaustausch-Algorithmus	203
13.6	Der Index-Kalkül	204
13.6.1	Unser Beispiel	205
13.6.2	Glatte Beziehungen	205
13.6.3	Matrix-Reduktion	205
13.6.4	Einzelne Logarithmen	208
13.6.5	Asymptotik	209
13.7	Schlüsselaustausch mit elliptischen Kurven	209
13.8	Schlüsselaustausch in anderen Gruppen	210
13.9	Wie schwierig ist das diskrete Logarithmusproblem?	211
13.10	Übungen	211
	Literatur	212
14	Elliptische Kurven Kryptographie	213
14.1	Einführung	213
14.1.1	Jacobianische Koordinaten	214

14.2	Elliptische Kurve Diskrete Logarithmen	215
14.3	Elliptische Kurven Kryptographie	215
14.4	Die Kosten von Operationen mit elliptischen Kurven	216
14.4.1	Verdoppelung eines Punktes	216
14.4.2	Von links nach rechts „Exponentiation“	217
14.5	Die Empfehlungen des NIST	219
14.6	Angriffe auf elliptische Kurven	221
14.6.1	Pohlig-Hellman Angriffe	221
14.6.2	Pollard Rho Angriffe	221
14.6.3	Pollard Rho für Kurven	223
14.6.4	Pollard Rho parallel.	224
14.7	Ein Vergleich der Komplexitäten	225
14.8	Übungen	226
	Literatur.	226
15	Gitterbasierte Kryptographie und NTRU	227
15.1	Quantencomputing.	227
15.2	Gitter: Eine Einführung	229
15.3	Harte Gitterprobleme	231
15.4	NTRU	232
15.5	Das NTRU-Kryptosystem	233
15.5.1	Parameter.	233
15.5.2	Schlüssel erstellen	234
15.5.3	Eine Nachricht verschlüsseln	234
15.5.4	Eine Nachricht entschlüsseln.	234
15.5.5	Warum das funktioniert.	236
15.5.6	Fehler bei der Entschlüsselung verhindern	236
15.6	Gitterangriffe auf NTRU	237
15.7	Die Mathematik des Gitterreduktionsangriffs	239
15.7.1	Andere Angriffe auf NTRU	240
15.7.2	Gitterreduktion	241
15.8	NTRU Parameterauswahl.	243
15.9	Übungen	243
	Literatur.	244
16	Homomorphe Verschlüsselung.	245
16.1	Einführung	245
16.2	Etwas Homomorphe Verschlüsselung	246
16.3	Vollständig Homomorphe Verschlüsselung	246
16.4	Ideale Gitter	247
16.5	Lernen mit Fehlern.	249

16.6	Sicherheit und homomorphe Auswertung von Funktionen	251
16.7	Eine entschuldigende Zusammenfassung	251
	Literatur.	252
	Ein tatsächlicher Erster Weltkrieg Chiffre.	253
	AES Code.	275
	Literatur.	305



Zusammenfassung

Der Wunsch von Regierungen, Generälen und sogar privaten Einzelpersonen, auf eine Weise zu kommunizieren, die verhindert, dass private Kommunikation von anderen gelesen wird, reicht Jahrtausende zurück. Die Fähigkeit Dritter, Nachrichten zu lesen, die nicht für sie bestimmt sind, oder dass Nachrichten für Dritte unlesbar sind, hat häufig den Lauf der Geschichte verändert. Der Einsatz von Technologie in den letzten 200 Jahren hat den Prozess der Nachrichtenübermittlung verändert, zunächst mit dem Telegraphen, dann mit drahtloser Kommunikation und jetzt mit dem Internet. Bei auf Drähten basierenden Telegraphensystemen benötigte ein Dritter physischen Zugang zum Kommunikationsmedium. Mit drahtlosem Radio wurde die Übertragung öffentlich und der Bedarf an sicheren Kommunikationen stieg. Bei Nachrichten, die in Paketen über das Internet gesendet werden, kann buchstäblich jeder von überall auf der Welt mithören. In diesem Kapitel werden wir kurz einige der Geschichte behandeln und wir werden grundlegende Begriffe und Anwendungen definieren, die im gesamten Buch fortgesetzt werden.

1.1 Geschichte

Das Problem der sicheren Kommunikation ist wahrscheinlich fast so alt wie die Zivilisation selbst. Bei der Übermittlung einer Nachricht an einen entfernten Korrespondenten, mit Gegnern irgendwo auf dem Weg, war es immer notwendig sicherzustellen, dass die Nachrichten von den Gegnern in der Mitte nicht verstanden werden können. In anderen Fällen wird Informationen nur zum Zweck der Privatsphäre (oder, wie man heute sagen könnte, zum Schutz des geistigen Eigentums) in einem Code geschrieben, der nur von den Eingeweihten gelesen werden kann. Leonardo da Vinci

schrieb zum Beispiel seine Notizen in Spiegelschrift, von rechts nach links. Einige haben vorgeschlagen, dass dies dazu diene, den Inhalt vor neugierigen Blicken zu schützen, obwohl eine andere Vermutung einfach ist, dass er Linkshänder war und durch das Schreiben von rechts nach links die Seite beim Schreiben nicht verschmierte.

Sichere Kommunikationen waren entscheidend für diejenigen, die Pläne schmiedeten und Intrigen spannten. Maria Stuart, Königin von Schottland, wurde zum Beispiel 1586 dabei erwischt, wie sie gegen Elisabeth von England plante. Wie sich herausstellte, war Elisabeths Kryptoanalytiker, Thomas Phelippes, besser in seinem Job als Marias Kryptograph, und verschlüsselte Nachrichten, die Maria mit dem Komplott gegen Elisabeth verbanden, wurden gelesen, was zur Hinrichtung Marias führte.

Thomas Jefferson entwarf eine Verschlüsselungsmaschine, die anscheinend nie zu seinen Lebzeiten gebaut wurde, aber im frühen 20. Jahrhundert weitgehend neu erfunden und als M-94 von der US-Armee verwendet wurde.

Einer der Klassiker der Literatur über Kryptographie, und zur damaligen Zeit ebenso umstritten wie er heute ein Klassiker ist, ist *Die amerikanische Schwarze Kammer*, von Herbert O. Yardley [1]. Yardley war ein Kryptograph während und nach dem Ersten Weltkrieg, arbeitete während des Krieges für die Armee-Geheimdienste und danach für ein gemeinsames schwarzes Kammer¹ Büro der Armee und des US-Außenministeriums, bis Außenminister Henry Stimson, berühmt sagte: „Gentlemen lesen nicht die Post der anderen“, beendete die Finanzierung und beendete damit effektiv das Büro. Eine der Hauptleistungen von Yardleys Büro erfolgte während der Verhandlungen der Washingtoner Flottenkonferenz von 1921–1922. Bei der Konferenz wurden die Verhältnisse der Schiffstonnage diskutiert, an die die Großmächte sich halten sollten, und entschlüsselte japanische Kommunikationen offenbarten die untere Grenze, die Japan zu der Zeit zu akzeptieren bereit war.

Während Yardley für die Armee und das Außenministerium arbeitete, waren die anderen herausragenden amerikanischen Kryptographen der Ära William und Elizabeth Friedman, Ehemann und Ehefrau. Sie arbeiteten zusammen in den Riverbank Laboratories, einer privat finanzierten Einrichtung in der Nähe von Chicago des wohlhabenden Geschäftsmannes George Fabyan, und schrieben viele der frühen Arbeiten über Kryptographie, die noch heute als Klassiker der modernen Literatur angesehen werden. William Friedman leitete den Signals Intelligence Service der Armee und spielte eine Schlüsselrolle beim Knacken von Purple, dem diplomatischen Code der Japaner. Zu den berühmten Nachrichten, die aus Purple entschlüsselt wurden, gehörte die siebzehnteilige Kommunikation an die japanische Botschaft am Tag vor dem Angriff auf Pearl Harbor. Die Nachricht auf den letzten Seiten deutete klar darauf hin, dass der Krieg kurz bevorstand, aber der letzte Teil der Nachricht erreichte die Washingtoner Beamten nicht rechtzeitig. Verschwörungstheorien bestehen fort, dass die

¹ Der Begriff *schwarze Kammer* ist eine Übersetzung des französischen Begriffs *cabinet noire*, der im späten 16. Jahrhundert in Frankreich eingeführt wurde.

Beamten in Wirklichkeit Bescheid wussten, aber Japan angreifen lassen wollten, um die öffentliche Meinung zugunsten des Kriegseintritts zu beeinflussen.

Die Codebrechung spielte eine entscheidende Rolle im Zweiten Weltkrieg. Die britische Arbeit in Bletchley Park (nach frühen Durchbrüchen von drei polnischen Mathematikern) an der deutschen Marinechiffre, codenamed Enigma, ist Gegenstand vieler Bücher und Filme [2]. Obwohl der jüngste Film, *The Imitation Game*, die meiste Aufmerksamkeit erhalten hat, ist mehr Hollywood als Fakt in dem Skript; der frühere Film *Enigma* ist der Wahrheit viel näher. Für das Entschlüsseln von Enigma und nachfolgenden deutschen Chiffren wurden die ersten elektronischen Computer, genannt Colossus, von den Briten gebaut, aber da ihre Existenz bis Mitte der 1970er Jahre geheim gehalten wurde, wird normalerweise der amerikanische ENIAC als der erste echte Computer angesehen.

Vielleicht der größte Triumph der Kryptoanalyse im Krieg im Pazifik war die Fähigkeit der amerikanischen Marinegruppe, angeführt von Joseph Rochefort in Hawaii, die japanischen Kommunikationen, die zur Schlacht bei Midway führten, zu entschlüsseln. Berühmt ist, dass Rochefort und sein Team einen der japanischen Marinecodes geknackt hatten und die Zeit und die Schlachtordnung der japanischen Flotte entschlüsselt hatten, aber den Ort nur durch die Codebuchstaben „AF“ kannten. Im Verdacht, dass es Midway war, und inmitten einer hitzigen Diskussion mit anderen Marinegeheimdienstoffizieren, wurde eine Nachricht an die Basis auf Midway per sicherem Unterseekabel gesendet. Der Basis auf Midway wurde aufgetragen, zurück nach Hawaii zu senden, im Klartext, damit es von den Japanern aufgegriffen würde, die (falsche) Nachricht, dass eine Entsalzungsanlage ausgefallen sei und es einen Mangel an Frischwasser gebe. Innerhalb von 24 h entschlüsselte Rocheforts Gruppe eine japanische Nachricht, dass „AF an Wasser mangelt“ und wusste somit, dass der Angriff auf Midway gerichtet sein würde. Nur sechs Monate nach Pearl Harbor verloren die Japaner vier Flugzeugträger und ihre Marineexpansion wurde gestoppt.

Die Kryptoanalyse spielte auch eine Schlüsselrolle bei den D-Day-Landungen in der Normandie. Die Informationsbeschaffung war auf dem europäischen Kontinent viel schwieriger, da Landlinien verwendet wurden und nicht die drahtlosen Kommunikationen von Enigma zu Schiffen auf See. Aber es gab deutsche drahtlose Kommunikationen mit einer Chiffre, die TUNNY genannt wurde, die geknackt worden war und deren Nachrichten entschlüsselt werden konnten. Und zusätzlich zu den direkten Informationen über deutsche Pläne und Positionen aus dem Lesen deutscher Chiffren hatten die Alliierten den Vorteil von Nebeninformationen. Der japanische Botschafter und der Militärattaché hatten Führungen entlang der Küstenverteidigungen am Ärmelkanal erhalten. Sie hatten ausführliche Berichte geschrieben, die per Funk nach Japan zurückgesendet wurden, mit einer Chiffre, die gebrochen worden war, so dass die Alliierten aus erster Hand Informationen über die deutschen Verteidigungen hatten.

Kryptographie war auch immer politisch. Regierungen wollen oft nicht nur die Fähigkeit, die Nachrichten anderer Länder oder Gruppen zu lesen; sie wollen die Nachrichten ihrer eigenen Bürger lesen, und gelegentlich haben sie dafür gesetzliche Gründe. Die

Gerichte haben noch nicht endgültig entschieden, ob man verpflichtet werden kann, ein Passwort preiszugeben, oder ob das eine verpflichtende Selbstbelastung darstellen würde, die in den USA durch den Fünften Zusatzartikel verboten ist. Verschlüsselung ist nützlich für Geschäftstransaktionen, aber einige dieser Transaktionen sind illegal. Der Versuch der USA, den CLIPPER-Chip zu schaffen, scheiterte in den frühen 1990er Jahren kläglich, und es gab mehrere Studien über die Vor- und Nachteile privater Verschlüsselung [3–5]. Große Technologieunternehmen haben sich geweigert, Hintertüren in die Sicherheit ihrer Verbraucherprodukte einzubauen, trotz wiederholter Bemühungen, Gesetze zu verabschieden, die solche Hintertüren vorschreiben [6].

1.2 Einführung

Wir beginnen mit einigen Begriffen. Der *crypt* Teil unserer Begriffe bedeutet *geheim*. Eine *ologie* ist eine Studie, also ist *Kryptologie* etymologisch die Studie der Geheimnisse. Wir beschränken uns auf die Studie der *Kryptographie*, wo das *graphie* *Schreiben* bedeutet, also bedeutet *Kryptographie* *geheimes Schreiben*. Der andere oft gesehene Begriff ist *Kryptanalyse*, den wir als Analyse eines kryptographischen Unterfangens verstehen werden. Wir werden die Begriffe Kryptographie und Kryptologie möglicherweise gleichsetzen, aber versuchen, Kryptanalyse für den Prozess des Angriffs auf ein kryptographisches System zu reservieren, um die verborgenen Nachrichten zu lesen.²

Bei der Verwendung eines kryptographischen Systems beginnt man mit der ursprünglichen zu sendenden Nachricht, die als *Klartext* bezeichnet wird. Durch die Verwendung des kryptographischen Systems wird Klartext in *Geheimtext* umgewandelt, den der Sender beabsichtigt, dass er von niemandem außer dem beabsichtigten Empfänger verstanden wird. Wir werden die Wörter *verschlüsseln* und *chiffrieren* für den Prozess der Umwandlung von Klartext in Geheimtext und die Wörter *entschlüsseln* und *dechiffrieren* für den Prozess der Umwandlung von Geheimtext zurück in Klartext synonym verwenden. Verschlüsselung und Entschlüsselung verwenden immer eine Art von *Schlüssel*. Bis vor kurzem waren Kryptosysteme symmetrisch, wobei der gleiche Schlüssel für die Verschlüsselung und für die Entschlüsselung verwendet wurde. Dies machte es notwendig, dass der Schlüssel eine Information ist, die nur dem Sender und dem Empfänger bekannt ist, und es machte die Bewahrung dieser Geheimhaltung von größter Wichtigkeit. Die moderne Kryptographie stützt sich nun auf asymmetrische Systeme, bei denen der Schlüssel zur Verschlüsselung nicht derselbe ist wie der Schlüssel zur Entschlüsselung; ein öffentlicher Schlüssel wird vom Sender verwendet, und ein privater Schlüssel, der nur dem Empfänger bekannt ist (und daher logistisch viel weniger wahrscheinlich kompromittiert wird), wird zur Entschlüsselung verwendet.

²Und wir werden niemals das falsche Wort „Krypto-Analyse“ verwenden, denn das würde stattdessen bedeuten, dass die Analyse heimlich durchgeführt wird, was etwas ganz anderes ist.

Ein kleines Stück vereinfachender Jargon ist auch nützlich. Wir gehen davon aus, dass der Sender den Geheimtext erstellt und dass der Geheimtext von einem Gegner abgefangen werden könnte. Offensichtlich kann der Gegner dann das Muster der Bits, das den Geheimtext darstellt, beobachten. Wir werden jedoch sagen, dass der Gegner (oder sogar der beabsichtigte Empfänger) den Geheimtext nur *lesen* kann, wenn der Leser den Geheimtext wieder in den ursprünglichen Klartext umwandeln kann.

Ein Teil der Kryptographie, obwohl wir hier nicht darauf eingehen werden, ist die Idee eines *Codebuchs*, das historisch oft ein Buch war, das Code-Sequenzen (oft fünfstellige Zahlen) für jedes Wort liefert, das im Klartext verwendet werden soll. Der Klartext wird von Text in eine Sequenz von Code-Wörtern umgewandelt, die klar übertragen werden. Die Geheimhaltung des codierten Textes beruht darauf, dass ein Gegner keine Kopie des Codebuchs erhalten kann und nicht in der Lage ist, Frequenzanalysen oder Kribbeln zu verwenden, um die verwendeten Codewörter zu erraten, *Kribbeln* ist der Begriff für das Erraten einer Übereinstimmung zwischen bekannten oder erwarteten Wörtern im Klartext (wie Wochentage oder Tageszeiten) und Teilen des Geheimtextes oder codierten Textes und die Verwendung dieser Vermutung, um zu versuchen, ob mehr der Nachricht auf diese Weise entschlüsselt/entschlüsselt werden kann.

Schließlich erwähnen wir die *Codierungstheorie*, eine weitere Disziplin, die nicht wirklich mit dem, was wir hier präsentieren, verwandt ist. Der Hauptzweck der Codierungstheorie besteht darin, den eindeutigen Empfang von übertragenen Nachrichten auch bei Vorhandensein von Verfälschungen zu ermöglichen. Vielleicht ist die einfachste Technik in der Codierungstheorie die eines Paritätsbits, das am Ende einer Bitfolge angehängt wird. Bei „gerader Parität“ würde ein zusätzliches 0- oder 1-Bit, je nach Bedarf, zu einer Bitfolge hinzugefügt, so dass alle Folgen eine gerade Anzahl von 1-Bits hätten. Eine Folge, die an ihrem Bestimmungsort mit einer ungeraden Anzahl von 1-Bits ankäme, wäre bekannt, dass sie bei der Übertragung verfälscht wurde. Die Forschung und Substanz der Codierungstheorie besteht darin, Codes zu finden und zu analysieren, die die maximale Fähigkeit zur eindeutigen Übertragung und zur Erkennung und möglicherweise zur Korrektur von Verfälschungen bieten, während die geringste Anzahl von Bits über das Minimum hinaus verwendet wird, um für jedes der übertragenen Symbole eindeutige Codes zu liefern. Wir können das meiste von dem, was wir auf Englisch benötigen, mit 8-Bit-ASCII codes machen, zum Beispiel, und ein einziges zusätzliches Paritätsbit würde es uns ermöglichen, Zeichen als verfälscht zu kennzeichnen, für die eine ungerade Anzahl von Bits bei der Übertragung umgeklappt wurden. Wir würden jedoch nicht wissen, welches Bit oder welche Bits falsch waren.

1.3 Warum wird Kryptographie verwendet?

Es ist sinnvoll, nach den verschiedenen Zwecken der Kryptographie zu fragen. Wir können vier grundlegende Arten identifizieren, wie kryptographische Funktionen in der modernen Welt verwendet werden. Viele davon existieren außerhalb der Welt der

Internetkommunikation und -transaktion, aber wir werden uns auf diese Anwendungen konzentrieren.

- Der offensichtliche Grund für die Verwendung von Kryptographie ist die Wahrung der *Vertraulichkeit*, das heißt, die Kommunikation zwischen zwei Parteien nur für die beiden beteiligten Parteien verständlich zu machen.
- In der Welt des Internets werden wichtige Dokumente wie Verträge und Immobilien-transaktionen über das Internet versendet, das nicht sicher ist. Varianten der gleichen Funktionen, die zur Aufrechterhaltung der Vertraulichkeit verwendet werden, können verwendet werden, um die *Integrität* der übertragenen Dokumente zu gewährleisten, so dass beispielsweise der Dollarwert eines Vertrags nicht von einem böartigen Abfänger geändert werden könnte, während das Dokument von einer Partei zur anderen gesendet wird. Kryptographie wird auch verwendet, um Datenbanken und Dateien zu verschlüsseln, so dass, wenn ein böartiger Akteur einen Laptop stiehlt oder unbefugten Zugang zu einem Desktop erhält, die Dateien für niemanden außer dem Eigentümer der Dateien verständlich wären.
- Neben der Frage der Integrität eines Dokuments steht die Frage der *Authentifizierung* der Identität einer Person am anderen Ende einer Internettransaktion. Kryptographische Funktionen können verwendet werden, um sicherzustellen, dass man authentifizieren kann, mit wem man kommuniziert.
- Die traditionelle Methode, um sicherzustellen, dass eine Person ihre Verpflichtung nicht abstreiten kann, war eine nasse Unterschrift auf einem Dokument. Da wir wissen, dass das Internet von böartigen Akteuren bevölkert ist, ist es notwendig, dass die *Nicht-Abstreitbarkeit* von übertragenen Dokumenten nicht möglich ist. Wir benötigen ein Analogon zu einer nassen Unterschrift.

Im Rahmen einer Diskussion über die Verwendungszwecke von Kryptographie können wir zwei grundlegende und oft komplementäre Arten unterscheiden, wie kryptographische Algorithmen verwendet werden. Wenn das Ziel darin besteht, sicherzustellen, dass der Inhalt einer Nachricht von niemandem außer dem beabsichtigten Empfänger gelesen werden kann, dann wird der Prozess darin bestehen, einen dem Abfänger unbekannten Klartext in einen Geheimtext zu konvertieren, den nur der beabsichtigte Empfänger lesen kann.

Die Ziele der Authentifizierung oder Nicht-Abstreitbarkeit arbeiten in die entgegengesetzte Richtung. Wenn ein Sender seine Identität authentifiziert, kann davon ausgegangen werden, dass jeder Empfänger oder Abfänger den Klartext kennt (oder zumindest einen Teil des Klartextes abschreiben kann). Der Geheimtext muss daher etwas sein, das nur der angenommene Sender hätte senden können. Dies wird oft mit einer *Hash-Funktion* kombiniert, die eine scheinbar zufällige kurze Signatur für ein Dokument erzeugt, die nur vom ursprünglichen Dokument hätte erzeugt werden können. Obwohl es Variationen in den Algorithmen für diese beiden unterschiedlichen Zwecke gibt, gibt es erhebliche Gemeinsamkeiten.

1.4 Verschlüsselungsmodi

Algorithmen für Kryptographie arbeiten auf unterschiedliche Weise, und manchmal sind die Unterschiede sehr wichtig.

Ein *Blockchiffre* ist ein Algorithmus, der auf einen Block von Bits auf einmal arbeitet und einen Block von Geheimtext aus einem Block von Klartext erzeugt. Zum Beispiel arbeitet AES mit 128-Bit-Blöcken von Klartext, um 128-Bit-Blöcke von Geheimtext zu erzeugen. Ein RSA-Kryptosystem mit einem 2048-Bit-Schlüssel würde wahrscheinlich mit Blöcken der Größe 2048 arbeiten.

In einer *elektronischen Codebuch* Implementierung würden die Blöcke unabhängig voneinander behandelt, wobei jedes Paar von Klartext/Geheimtext-Blöcken unabhängig voneinander ist. Dies kann in vielen Fällen unsicher sein. AES hat zum Beispiel nur einen 16-Byte-Block, und wenn der zu verschlüsselnde „Text“ ein Bild ist, könnten große Teile des Bildes den gleichen Hintergrund-Klartext haben und würden daher als der gleiche Geheimtext verschlüsselt. Dies kann die Konturen des zugrunde liegenden Bildes erkennbar machen.

Um zu verhindern, dass identische Klartextblöcke in identische Geheimtextblöcke verschlüsselt werden, kann man *Blockchiffre-Verkettung* implementieren. Dabei wird ein Initialisierungsvektor aus zufälligen Bits mit dem ersten Block des Klartexts XOR-verknüpft, der dann verschlüsselt wird. Der Geheimtext des Blocks n wird dann so verwendet, als ob er ein Initialisierungsvektor für den Block $n + 1$ wäre, bevor dieser Block verschlüsselt wird. Durch diese Vorgehensweise werden selbst identische Blöcke des Originaltextes durch ein Bitmuster, das zufälliger ist, modifiziert, bevor die Blöcke verschlüsselt werden, und es ist nicht der Fall, dass man den gleichen Geheimtext aus den gleichen Blöcken von identischem Klartext erzeugt.

Im Gegensatz zu Blockchiffren wird bei einem *Stromchiffre* eine Sequenz von „zufälligen“ Bits mit einem Strom von Klartextbits XOR-verknüpft, um den Geheimtext zu erzeugen. Dies erfordert natürlich, dass man einen deterministischen Algorithmus hat, der sowohl dem Sender als auch dem Empfänger bekannt ist und einen Strom von Bits erzeugt, der für alle praktischen Zwecke zufällig erscheint. Stromchiffren wurden oft mit linearen Rückkopplungsschieberegistern konstruiert, zum Beispiel, wie in Kap. 6 diskutiert wird.

Wir bemerken, dass mit dem digitalen Zeitalter eine Standardisierung gekommen ist. Im Gegensatz zu einer Nachricht, die ausschließlich auf Papier gesendet wird, werden alle elektronisch gesendeten Nachrichten als codierte Nullen und Einsen gesendet, und somit können alle Nachrichten als Zahlen behandelt werden. In einem Computer gespeicherter Text wird normalerweise als Unicode-Zahlen gespeichert, die die zu speichernden Zeichen darstellen. Unicode ist die mehrbyte Erweiterung zu ASCII (American Standard Code for Information Interchange), das einbyte (ursprünglich nicht alle Bits verwendend) Codes für Buchstaben, Zahlen, Interpunktion und Steuerzeichen zuwies. Solche Codierungen begannen mit Samuel F. B. Morse und Emile Baudot im 19. Jahrhundert für die Verwendung mit Telegraphen. Morse-Code ist variabler Länge, wobei die häufigeren Buchstaben kürzere Codewörter bekommen. Baudot und nachfolgende Codes vor

Unicode waren feste Längencodes. Unicode, noch in Arbeit, wurde entwickelt, um Codes für alle Alphabete und diakritischen Zeichen aller Sprachen der Welt sowie für solche neuen Dinge wie Emojis bereitzustellen. Unicode ist variabler Länge: die ursprünglichen Codes, die von ASCII abgeleitet sind, sind einbyte. Wenn mehr Bytes benötigt werden (da ein einzelnes Byte nur 256 verschiedene Werte codieren kann), hat das erste Byte ein Signal codiert, das anzeigt, dass das nächste Byte eine Fortsetzung des ersten ist. Drei-Byte-Codes machen diese Fortsetzung einfach zweimal.

Die moderne Kryptographie nimmt dann eine von zwei verschiedenen Formen an. In den meisten Public-Key-Kryptographien werden die Bitmuster des zugrunde liegenden Textes als die binäre Zahl betrachtet, die das Muster darstellt, und Berechnungen werden auf dieser Zahl durchgeführt. In Systemen wie AES, dem Advanced Encryption Standard, werden die Bitmuster durch Umordnung und Anwendung von binären Funktionen wie einem XOR zufällig erscheinen gemacht.

1.5 Angriffsmodi

Bei der Kryptoanalyse gibt es vier grundlegende Arten von Angriffen, die durchgeführt werden können.

- Bei einem *Geheimtext-only-Angriff* werden uns nur Geheimtexte präsentiert.
- Bei einem *bekannten Klartext-Angriff* kennen wir den entschlüsselten Klartext und wir haben den entsprechenden Geheimtext. Unser Ziel hier ist es, den Verschlüsselungsalgorithmus zu entdecken, wenn wir ihn noch nicht kennen, und den Schlüssel, wenn wir den Verschlüsselungsalgorithmus durch andere Methoden gefunden haben.
- Bei einem *gewählten Geheimtext-Angriff* ist es möglich, Informationen darüber zu gewinnen, wie man entschlüsselt, indem man die Entschlüsselungen von gewählten Instanzen von Geheimtext hat.
- Bei einem *gewählten Klartext-Angriff* kann der Kryptoanalytiker den Klartext wählen und daraus den Geheimtext für diesen Klartext erhalten. Bei Public-Key-Verschlüsselungsschemata zum Beispiel ist die Möglichkeit, eine Nachricht zu verschlüsseln, für jeden möglich, der Klartext hat, weil der Verschlüsselungsschlüssel selbst öffentliches Wissen ist.

1.6 Wie viele Sekunden hat ein Jahr?

Bei der Messung der Brute-Force-Komplexität eines Kryptosystems muss man oft eine grobe Schätzung für die benötigte Zeit berechnen, um das System mit einem völlig naiven Angriff zu knacken. Das Data Encryption Standard (DES) Kryptosystem der Vereinigten

Staaten hatte zum Beispiel einen 56-Bit-Schlüssel. Es gab also $2^{56} \approx 7,2 \times 10^{16}$ mögliche Schlüssel. Würde man DES mit Brute-Force angreifen, würde man einfach jeden Schlüssel nacheinander testen. Bei einer groß angelegten Operation mit vielen möglichen Nachrichten zum Entschlüsseln würden wir im Durchschnitt nur bis zur Hälfte des Schlüsselraums gehen, bevor wir auf den richtigen Schlüssel stoßen, also würden wir im Durchschnitt erwarten, dass wir $2^{55} \approx 3,6 \times 10^{16}$ mögliche Schlüssel testen müssen, bevor wir eine Nachricht entschlüsseln können.

Wie lange würde das dauern? Wie einmal gesagt wurde, ist ein Test, ob ein Sprecher wirklich Supercomputing betrieben hat, ob er weiß, wie viele Sekunden es in einem Jahr gibt. Es gibt 86,400 s an einem Tag und $31,536,000 \approx 3 \times 10^7$ s, oder 3×10^{16} ns, in einem Nicht-Schaltjahr. Bei groben Berechnungen wie dieser in Binär stellen wir auch fest, dass $2^{25} = 33,554,432$, also kann man das für grobe Rückseitenberechnungen, die in Binär einfacher zu machen sind, als die Anzahl der Sekunden in einem Jahr verwenden.

Moderne Computer laufen mit etwa einer 3 GHz Uhr, was bedeutet, dass jede Taktperiode etwa 0,33 ns beträgt. Aber moderne Computer sind stark gepipelt, so dass es nicht der Fall ist, dass Anweisungen nur 0,33 ns zum Abschließen benötigen. Bequemerweise sagen wir, dass wir einen einzelnen DES-Schlüssel in 1 ns testen können, was bedeutet, dass wir 10^9 Schlüssel jede Sekunde testen können. Mit dieser Rate könnten wir den DES-Schlüsselraum in $3,6 \times 10^7$ s erschöpfen, was etwas über ein Jahr ist.

Erst jetzt müssen wir uns Sorgen machen, weniger schlampig in unseren Schätzungen zu sein. Etwas über ein Jahr liegt im Bereich eines konzertierten Angriffs. Da das Testen von Schlüsseln peinlich parallel ist, könnten wir verschiedene Teile des Schlüsselraums verschiedenen Computern übergeben und die gesamte Berechnung parallel durchführen. Mit 1000 Computern, bei 10^9 pro Sekunde auf jedem Computer getesteten Schlüsseln, ist unsere Brute-Force-Zeit auf etwa 36,000 s reduziert, was etwa zehn Stunden entspricht. Und jetzt fangen wir an, weniger schlampig zu sein. Ein Schlüssel pro Nanosekunde? Das ist wahrscheinlich optimistisch, aber wenn wir nur um den Faktor 5 daneben liegen, dann finden wir immer noch etwa alle zwei Tage Schlüssel. Können wir 1000 Computer bekommen? Wenn wir ein großer Player in der Computerwelt sind, ist das nicht unvernünftig, vorausgesetzt, dass das Entschlüsseln von Nachrichten für die Mächtigen wichtig ist. Das Anpassen der Schätzungen nach oben und unten führt immer noch zu dem grundlegenden Schluss, dass DES in einer vernünftig kleinen Anzahl von Tagen mit Brute-Force geknackt werden kann.

Wir können DES mit AES vergleichen. Da AES einen 128-Bit-Schlüssel hat und $2^{128} \approx 3,4 \times 10^{38}$, wissen wir, dass ein Brute-Force-Angriff nicht funktionieren wird, egal was wir tun. Mit 1000 Computern, die 10^9 Schlüssel pro Sekunde testen, und 30 Millionen Sekunden in einem Jahr, würde es bis zur Hälfte des Schlüsselraums

$$\frac{1,7 \times 10^{38}}{10^3 \times 10^9 \times 3,0 \times 10^7} \approx 5 \times 10^{18}$$

Tab. 1.1 Brute-Force-Zeit zum Knacken, für verschiedene Bitlängen

Schlüsselgröße	Jahre	Schlüsselgröße	Jahre
56	1,14	320	$3,39 \times 10^{79}$
64	292,47	352	$1,45 \times 10^{89}$
96	$1,26 \times 10^{10}$	384	$6,25 \times 10^{98}$
128	$5,40 \times 10^{21}$	416	$2,68 \times 10^{108}$
160	$2,32 \times 10^{31}$	448	$1,15 \times 10^{118}$
224	$4,27 \times 10^{50}$	480	$4,95 \times 10^{127}$
256	$1,84 \times 10^{60}$	512	$2,13 \times 10^{137}$
288	$7,89 \times 10^{69}$		

Jahre dauern, um einen Schlüssel zu finden. Und jetzt hilft es nicht, die Ungenauigkeiten in unseren Schätzungen zu korrigieren. Wenn wir 100 Mal so viele Computer haben, die jeweils 100 Mal schneller sind, wird der Exponent nur auf 14 reduziert. Die Erde existiert erst seit etwa $4,5 \times 10^9$ Jahren; und unser Brute-Force-Angriff würde immer noch etwa 100.000 Mal länger dauern als das.

Um die Dinge ins rechte Licht zu rücken, geben wir in Tab. 1.1 die erwartete Zeit für einen Brute-Force-Angriff auf ein Kryptosystem mit unterschiedlichen Schlüssellängen an. Dies geht davon aus, dass wir einen Schlüssel in einer Nanosekunde testen und dass wir im Durchschnitt nur die Hälfte der Schlüssel testen müssen, um Erfolg zu haben. Die Lehre ist klar; Brute Force wird nicht funktionieren.

1.7 Kerckhoffs' Prinzip

Schließlich zitieren wir das Prinzip, das der niederländische Kryptograph Auguste Kerckhoff im 19. Jahrhundert formulierte: Ein Kryptosystem sollte sicher sein, auch wenn alles über das System bekannt ist, außer dem Schlüssel, der zum Verschlüsseln einer bestimmten Nachricht verwendet wird.

Claude Shannon wiederholte das 1949 auf eine andere Weise und stellte fest, dass man in der Kommunikation davon ausgehen muss, dass der Feind alles über das System weiß.

Diese Prinzipien scheinen selbstverständlich. Der kontrastierende Standpunkt ist das, was abwertend als „Sicherheit durch Obskurität“ bezeichnet wird. Unternehmen haben oft natürlich Geschäftsgeheimnisse. Patente sind Mechanismen, durch die Erfindungen, die für den Erfinder von Vorteil sind, öffentlich gemacht werden, aber nicht ohne Lizenzierung verwendet werden können. Es ist eine allgemeine Regel in der Sicherheit, und nicht weniger für die Kommunikationssicherheit, dass man nicht davon ausgehen kann, dass Geheimnisse in der Gestaltung eines Systems geheim bleiben. Wenn

es etwas von Wert zu finden gibt, ist der konservative Ansatz zur Sicherheit davon auszugehen, dass Versuche unternommen werden, das aufzudecken, was Wert hat. Und der konservative Ansatz besteht darin, anzunehmen, dass selbst wenn externe Spione nicht eindringen können, Insider, die Bescheid wissen, korrumpiert oder erpresst werden könnten. Wie Benjamin Franklin es ausdrückte: „Drei können ein Geheimnis bewahren, wenn zwei von ihnen tot sind.“

1.8 Übungen

Präsentieren Sie eine Analyse, unter Verwendung seriöser Quellen, über die Bedeutung der Kryptographie in Bezug auf Folgendes:

1. Maria Stuart, Königin von Schottland.
2. Der Washingtoner Flottenvertrag.
3. Der Angriff auf Pearl Harbor.
4. Die Verwendung von Enigma-Entschlüsselungen in der Schlacht im Atlantik im Zweiten Weltkrieg.
5. Die Verwendung von Entschlüsselungen in der Schlacht um Midway.
6. Die Venona-Entschlüsselungen und die Prozesse und Hinrichtung von Julius und Ethel Rosenberg.
7. Die Arbeit von William und Elizabeth Friedman in den Riverbank Laboratories.
8. Die Arbeit von Elizabeth Friedman während der Prohibitionszeit in den Vereinigten Staaten.
9. Die Kontroverse über die Veröffentlichung der RSA-Verschlüsselungsmethode durch Rivest, Shamir und Adleman Ende der 1970er Jahre.
10. Der in den 1990er Jahren von der US-Regierung vorgeschlagene CLIPPER-Chip.
11. Die aktuelle Kontroverse darüber, ob Personen gezwungen werden können, Informationen auf Laptop-Festplatten zu entschlüsseln, wenn sie die Grenze in ein anderes Land überqueren.
12. Die aktuelle Kontroverse darüber, ob Technologieunternehmen dazu verpflichtet werden sollten, eine Hintertür in die Sicherheit und Kryptographie einzubauen, damit Strafverfolgungsbehörden Zugang zu mit einem Mobiltelefon verbundenen Informationen erhalten können.

Literatur

1. H.O. Yardley, *The American Black Chamber* (Bobbs-Merrill, Indianapolis, 1931), S. 140–171
2. G. Welchman, *The Hut Six Story* (McGraw-Hill, New York, 1982).
3. W. Hollingsworth, H. Sachs, A.J. Smith, The CLIPPER processor: instruction set architecture and implementation. *Commun. ACM* **32**, 200–219 (1989)

4. S. Landau, S. Kent, C. Brooks, S. Charney, D. Denning, W. Diffie, A. Lauck, D. Miller, P. Neumann, D. Sobel, Crypto policy perspectives. *Commun. ACM* **37**, 115–121 (1994)
5. S. Landau, S. Kent, C. Brooks, S. Charney, D. Denning, W. Diffie, A. Lauck, D. Miller, P. Neumann, D. Sobel, *Codes, Keys, and Conflicts: Issues in U.S. Crypto Policy*. Association for Computing Machinery Report (1994)
6. H. Abelson, R. Anderson, S.M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, M. Green, P.G. Neumann, S. Landau, R.L. Rivest, J.I. Schiller, B. Schneier, M. Specter, D.J. Witzner, Keys under doormats: mandating insecurity by requiring government access to all data and communications. *Commun. ACM* **58**, 24–26 (2015)



Zusammenfassung

Bis zum Computerzeitalter war das Erstellen und Brechen von Chiffren eine Aufgabe, die extreme Konzentration und Sorgfalt erforderte. Suchbäume basierend auf Vermutungen können auf Computern programmiert und mit hoher Geschwindigkeit ausgeführt werden, wobei wir die Geschwindigkeit des Computers und die Leichtigkeit der Datenverfolgung in Datenstrukturen nutzen können, um uns nicht allzu sehr um die Verfolgung von Pfaden mit geringer Wahrscheinlichkeit zu kümmern. Die Kosten in Zeit und Aufwand für die Suche mit Bleistift und Papier hätten viel bessere Vermutungen über den richtigen Pfad durch den Baum erfordert. Die Kryptoanalyse in der ersten Hälfte des zwanzigsten Jahrhunderts erforderte Kenntnisse von Sprachmustern und Frequenzstatistiken, und sowohl die Verschlüsselung als auch die Entschlüsselung mussten Prozesse sein, die leicht erinnert und befolgt werden konnten. In diesem Kapitel werden wir einige klassische Chiffren beschreiben (die leicht mit einem Programm auf einem Desktop-Computer angegriffen werden könnten) sowie einige statistische Eigenschaften von Sprachen, die verwendet werden könnten, um diese nun veralteten Chiffren anzugreifen. Es gibt zwei grundlegende Formen einfacher Chiffren. Bei einer *Substitutionschiffre* ersetzt man für jeden Buchstaben im zugrunde liegenden Alphabet ein anderes Symbol (vielleicht einen anderen Buchstaben im selben Alphabet, oder manchmal ein ganz anderes Symbol). Bei einer *Transpositionschiffre* bleiben die Buchstaben des zugrunde liegenden Alphabets gleich, aber ihre Reihenfolge wird in eine andere Reihenfolge transponiert. Dabei kann man den Begriff „Buchstabe“ als einzelnen Buchstaben oder vielleicht als Paar von Buchstaben verstehen. Wir unterscheiden von Anfang an ein *Codebuch* von einer

Chiffre, obwohl die beiden eng miteinander verbunden sein können. Traditionelle Codebücher waren eine Form der Geheimhaltung von Kommunikation, indem für jedes der einzelnen Wörter in der Nachricht eine feste Länge (oft fünf) Sequenz von Zahlen ersetzt wurde. Man kann sich ein solches Codebuch als Substitutionschiffre vorstellen, bei der die Symbole Wörter (natürlich von variabler Länge) sind, für die man numerische Symbole ersetzt. Wir werden auch nur kurz (genau hier) den Begriff der *Steganographie* erwähnen, bei der eine Nachricht in einer scheinbar harmlosen Kommunikation versteckt ist. Eine Version davon wäre ein Brief, in dem die versteckte Nachricht die Sequenz der ersten Buchstaben der Wörter des Textes ist. Eine modernere umgekehrte Version der Steganographie ist das *digitale Wasserzeichen*, bei dem ein digitales Muster in ein Dokument, normalerweise ein Bilddokument, eingefügt wird, so dass die Herkunft des Bildes authentifiziert werden kann, wenn es illegal ohne Zuschreibung oder Lizenzgebühr entnommen wird. Dies ist nicht unähnlich der scheinbaren Einbeziehung von absichtlichen Fehlern in Karten, sagen wir, so dass der Inhaber des Urheberrechts der Karte argumentieren könnte, dass die Karte illegal kopiert wurde. Der Autor wünscht sich sehr, dass er die Straßenkarte von Louisiana (wo er aufgewachsen ist) behalten hätte, die eine Straße südlich von Venice, Louisiana, und eine Brücke über den Mississippi nach Pilottown zeigt. Eine solche Straße oder Brücke hat es nie gegeben; Pilottown ist der Ort, an dem die Mississippi River Piloten die ankommenden Schiffe treffen und das Ruder auf dem Weg flussaufwärts zum Hafen von New Orleans übernehmen, und wo sie auf der Ausreise das Ruder an die seefahrenden Piloten übergeben. Die „Stadt“ kann nur per Wasser erreicht werden; es gibt keine Straße südlich von Venice und keine Brücke über den Mississippi.

2.1 Substitutionschiffren

2.1.1 Caesar-Verschlüsselungen

Vielleicht die einfachste und historischste der Substitutionschiffren ist die, die Julius Caesar zugeschrieben wird. Die klassische Caesar-Verschlüsselung ist eine Verschiebung, modulo 26, der Buchstaben des Alphabets:

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

Mit anderen Worten, jeder Buchstabe des Klartextes wird einfach um drei Buchstaben nach unten verschoben, um den verschlüsselten Text zu erzeugen.

Wir könnten sehen

Klartext	i	a	m	t	h	e	m	e	s	s	a	g	e
Geheimtext	l	d	p	w	k	h	p	h	v	v	d	o	h

2.1.2 Zufällige Substitutionen

Die Caesar-Verschlüsselung hat den großen Vorteil, dass sie leicht zur Verschlüsselung einer Nachricht verwendet werden kann. Man muss nur wissen, dass die Verschlüsselung ein Verschieben um drei Buchstaben nach unten bedeutet. Andererseits ist das Muster „jeder Buchstabe verschiebt sich um 3“ eine Schwäche in der Verschlüsselung. Sobald ein Kryptoanalytiker feststellt, dass mehrere Buchstaben anscheinend um drei nach unten verschoben wurden, könnte er vermuten, dass dies für alle Buchstaben zutrifft, und die Verschlüsselung wäre gebrochen.

Betrachten Sie stattdessen eine Substitution, bei der eine Permutation der $26! \approx 15 \times 10^{24}$ möglichen Permutationen der 26 Buchstaben des englischen Alphabets als Verschlüsselungsmechanismus gewählt wird. Mit einer zufällig gewählten Permutation gäbe es kein Muster von „Verschiebung um drei“, das offensichtlich wäre, und die Verschlüsselung wäre viel schwerer zu brechen.

2.1.3 Vigenère als Beispiel für polyalphabetische Substitutionen

Sowohl die Caesar-Verschlüsselung als auch eine zufällige Substitutionsverschlüsselung verwenden ein einzelnes Alphabet zur Verschlüsselung von Symbolen. Die Vigenère-Verschlüsselung, die erstmals von Giovan Battista Bellaso beschrieben und später fälschlicherweise Vigenère zugeschrieben wurde, ist eine *polyalphabetische Verschlüsselung*, bei der (wie der Name schon sagt) mehrere Alphabete verwendet werden, um einen Satz von Symbolen durch einen anderen zu ersetzen.

Wir beginnen mit einer Tabelle von Symbolen, die für jede der möglichen Verschiebungen einer Caesar-ähnlichen Verschlüsselung verschoben wurden.

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

Wir verwenden jetzt ein leicht zu merkendes Wort als Schlüssel, sagen wir „buell“, und wiederholen es über die gesamte Nachricht:

Schlüssel	b u e l l b u e l l b u e
Klartext	i a m t h e m e s s a g e
Geheimtext	j v q f t f g i e e b b i

Um zu verschlüsseln, beginnen wir mit dem ersten Buchstaben, „i“, und dem ersten Buchstaben des Schlüssels, „b“. Wir können die Spalte „i“ und die Zeile „b“ nehmen und finden den Chiffrebuchstaben „j“. Dann nehmen wir den zweiten Buchstaben und den Schlüsselbuchstaben, und unter der Spalte „a“ für die Zeile „u“ finden wir den Chiffrebuchstaben „v“. Und so weiter. Die Entschlüsselung ist genau das Gegenteil.

2.2 Spracheigenschaften und -muster

Bevor wir zu Transpositionsschiffren und dann zur Kryptoanalyse dieser einfachen Chiffren übergehen, lohnt es sich, einen kleinen Umweg zu machen und die Beobachtung (und Mathematik) von Mustern, die in jeder menschlichen Sprache auftreten, zu betrachten.

2.2.1 Buchstabenhäufigkeit

Das Erste, was zu beachten ist, ist, dass die Buchstaben nicht gleich oft im Text erscheinen, und dann, dass die Buchstabenhäufigkeit ein starker Indikator für die Sprache ist. Wir präsentieren in Tab. 2.1 drei Häufigkeitsprozentsätze für Englisch. Die erste Spalte stammt von einer Website der Cornell University [1] und die zweite und dritte von einer Häufigkeitszählung der Gutenberg-Projekt [2] Versionen von Charles Dickens' *David Copperfield* und Charles Darwins *Reise der Beagle*. Wir haben Abschnitte abgegrenzt, in denen die Buchstaben in einem Block gleich sind, aber die Häufigkeiten leicht unterschiedlich sind; nur im vorletzten Block, der die Buchstaben u, c, m, f, y, w, g und p enthält, sehen wir einen signifikanten Unterschied in der Reihenfolge.

Für kryptanalytische Zwecke können solche Häufigkeitszählungen fast sofort einen Substitutionsschiffre von einem Transpositionsschiffre unterscheiden. Da ein Transpositionsschiffre die Häufigkeit der Buchstaben im Text nicht ändert, sollte die Häufigkeitszählung in etwa der gleichen sein wie eine Benchmark-Häufigkeitszählung. Im Falle eines Substitutionsschiffres könnte man fast sofort vermuten, dass der häufigste Buchstabe der Ersatz im Englischen für den Buchstaben „e“ ist.

Im Falle eines Transpositionsschiffres können die Häufigkeiten der Buchstaben dazu dienen, die zugrunde liegende Sprache zu identifizieren, zumindest für die großen europäischen Sprachen.

Eine ähnliche Häufigkeitszählung bei den Wörtern selbst kann auch zum Raten verwendet werden, insbesondere wenn ein Codebuch verwendet wird. Die acht häufigsten Wörter in *David Copperfield*, zum Beispiel, sind in der Reihenfolge, „the“, „I“, „and“, „to“, „of“, „a“, „in“, und „my“, die zusammen fast 21% der Gesamtwörter im Text ausmachen. Aus diesem Grund wurden die meisten Codebücher mit mehreren Codewörtern für die sehr häufigen Wörter erstellt, um die Häufigkeiten, mit denen sie vorkommen, zu verbergen.

Tab. 2.1 Drei Häufigkeitsverteilungen für englischen Text

Cornell		Dickens		Darwin	
e	12.02	e	12.08	e	13.06
t	9.10	t	8.84	t	9.23
a	8.12	a	8.17	a	8.36
o	7.68	o	7.74	o	7.23
i	7.31	i	7.24	n	6.87
n	6.95	n	6.84	i	6.80
s	6.28	h	6.06	s	6.63
r	6.02	s	6.05	r	6.27
h	5.92	r	5.75	h	5.87
d	4.32	d	4.70	l	4.08
l	3.98	l	3.79	d	3.95
u	2.88	m	3.15	c	2.95
c	2.71	u	2.83	u	2.62
m	2.61	w	2.60	f	2.57
f	2.30	y	2.26	m	2.28
y	2.11	c	2.25	w	2.05
w	2.09	f	2.17	g	1.93
g	2.03	g	2.10	p	1.77
p	1.82	p	1.70	y	1.57
b	1.49	b	1.52	b	1.65
v	1.11	v	0.93	v	1.13
k	0.69	k	0.90	k	0.55
x	0.17	x	0.14	x	0.18
q	0.11	j	0.10	q	0.13
j	0.10	q	0.09	z	0.10
z	0.07	z	0.02	j	0.08

2.2.2 Wortgrenzen

Wir bemerken, dass es üblich war, bei Substitutions- oder Transpositionsschiffren die Leerzeichen zwischen den Wörtern wegzulassen und den Text zusammenzuführen. Englisch hat zum Beispiel nur zwei Wörter mit einem Buchstaben, „a“ und „I“, und relativ wenige Wörter mit zwei oder drei Buchstaben. Dies macht das Raten viel einfacher.^{1,2}

¹ Im Film „Sneakers“ von 1992 findet die Kryptanalyse auf einem Computerbildschirm statt, mit leeren Räumen, die Wörter trennen. Dies war die Ursache für einige ernsthafte Verachtung unter den verschiedenen Kryptologen meiner Bekanntschaft.

² Wir geben sicherlich zu, dass im Klartext so etwas wie „der Buchstabe b“ erscheinen könnte, aber diese sind selten.

2.2.3 Cribbing

Einfache Chiffren, wenn sie schlecht gemacht sind, können extrem unsicher sein. Das Folgende ist eine Version einer Chiffre, die der Autor tatsächlich aufgefordert wurde zu entschlüsseln. Der Brief war als Brief formatiert, komplett mit Interpunktion und Leerzeichen zwischen den Wörtern. Die „zusätzliche Information“ war, dass bekannt war, dass dies ein Brief vom Absender an einen Empfänger mit einer bekannten romantischen Verbindung war.

```
pase zhic,
xqasta fchy mism u qhva dhj scp mism u kutt dhj s qhm.
qhva,
ksemis
```

Es ist nicht schwer, dies zu cribben. Wie beendet man einen Brief mit einer romantischen Verbindung? Mit dem Wort „love“, natürlich, also cribben wir qhva zu love. Das gibt uns Vermutungen für zwei Vokale, „o“ und „e“. Wir können „u“ und „s“ zu „a“ und „i“ oder umgekehrt raten. Danach ist es einfach, selbst bei einer Nachricht dieser Kürze.

Die Lehre ist, dass man keine Kommunikationssicherheit erwarten kann, wenn man Informationen, die nicht Teil des eigentlichen Chiffresystems sind, aber abgeleitet, geraten oder gecribbt werden können, nicht verschleiert.

2.2.4 Entropie

Das Ausmaß, in dem eine natürliche Sprache zugrunde liegende Muster hat, kann mathematisch gemacht werden, mit dem Konzept der *Entropie* in der Informationstheorie, entwickelt von Claude Shannon und brillant beschrieben in Hamming's Buch [3]. Angenommen, wir haben „Text“ als eine Sequenz von „Symbolen“ (die wir in diesem Fall als Buchstaben nehmen), ist die Idee folgende. Wir wollen eine mathematische Funktion, die das Ausmaß misst, in dem ein neues Symbol mehr „Information“ liefert. Zum Beispiel liefert im Englischen das Sehen eines „u“ nach einem „q“ relativ wenig Information, außer der Tatsache, dass man anscheinend ein englisches Wort liest. Etwas anderes als ein „u“ nach einem „q“ zu sehen, liefert Informationen, da es darauf hinweist, dass das Wort wahrscheinlich nicht Englisch ist.

2.2.4.1 Information

Wir gehen davon aus, dass wir ein Alphabet von Symbolen haben

$$\{a_1, \dots, a_k\}$$

jedes davon erscheint im Text mit festen Wahrscheinlichkeiten

$$p_i = p(a_i),$$

und wir nehmen an, dass diese den Raum abdecken:

$$\sum_{i=1}^k p_i = 1$$

Wir möchten die Menge an Information $I(p)$, oder Überraschung, beim Sehen eines Symbols a messen, das mit Wahrscheinlichkeit p auftritt. Wir nehmen drei Eigenschaften der Funktion $I(p)$ an.

1. Information ist immer nichtnegativ, das heißt, $I(p) \geq 0$.
2. Information ist additiv für unabhängige Ereignisse, das heißt,

$$I(p_i p_j) = I(p_i) + I(p_j).$$

3. $I(p)$ ist eine stetige Funktion von p .

Wir können nun bestimmen, welche Arten von Funktionen $I(p)$ diese Eigenschaften haben.

Zuerst stellen wir fest, dass wir haben müssen

$$I(pp) = I(p^2) = I(p) + I(p) = 2I(p).$$

Aus diesem können wir rekursiv erweitern, um zu bekommen

$$I(p^n) = I(pp^{n-1}) = I(p) + I(p^{n-1}) = I(p) + (n-1)I(p) = nI(p).$$

Da $I(p)$ stetig ist, können wir $q = p^n$ einsetzen, so $p = q^{1/n}$ und somit

$$I(q) = nI(q^{1/n}).$$

und dann wenden wir das oben genannte Argument unter Verwendung der zweiten Bedingung an, um zu bekommen

$$I(p^{m/n}) = (m/n)I(p).$$

Wir schließen daraus, dass $I(p)$ für rationale Zahlen sich genau wie ein Logarithmus verhält. Da wir angenommen haben, dass $I(p)$ stetig ist, können wir von allen rationalen Zahlen auf alle reellen Zahlen erweitern, und somit dass

$$I(p) = r \log p.$$

Jetzt, da p im Bereich von 0 bis 1 liegt, wissen wir, dass $\log p$ negativ ist, und somit muss r negativ sein, damit die erste Eigenschaft gilt. Betrachten wir die zweite Eigenschaft, dass $I(p)$ additiv ist, haben wir dass

$$I(p_1 \dots p_m) = I(p_1) + \dots + I(p_m) = r(\log p_1 + \dots + \log p_m).$$

Wir können r wählen, wie wir wollen, ohne das Verhalten der Funktion $I(p)$ zu ändern; verschiedene Werte von r würden nur die absoluten Zahlen skalieren, ohne irgendeinen

relativen Unterschied zu ändern, daher gibt es keinen Grund, einen Wert von r anders als -1 in Betracht zu ziehen. Mit diesem haben wir

$$I(p) = -\log p = \log(1/p),$$

und dies ist die Standardfunktion, die zur Messung der Informationsmenge verwendet wird.

2.2.4.2 Entropie

Wir können die Informationsfunktion verwenden, um ein Maß für die durchschnittliche Information zu liefern, die man erhält, wenn man Symbole aus einem festen Alphabet liest. Wenn die Symbole

$$\{a_1, \dots, a_k\}$$

aus einem Alphabet S im Text mit festen Wahrscheinlichkeiten

$$p_i,$$

erscheinen, erhält man

$$p_i \log(1/p_i)$$

durchschnittlich Einheiten von Information aus dem Symbol a_i . Summiert man über alle Symbole, erhält man

$$H(S) = \sum_{i=1}^k p_i \log(1/p_i)$$

als die durchschnittliche Information, die ein durchschnittlicher Text über das Alphabet S vermittelt. Wir nennen dies die *Entropie* des Alphabets oder des Textes, der mit diesem Alphabet geschrieben wurde.

2.2.4.3 Beispiele

Betrachten Sie, was passiert, wenn eine faire Münze geworfen wird, und stellen Sie sich vor (da die Berechnung in einer binären Welt erfolgt), dass ein Ausgang von Kopf einer 1 Bit entspricht und ein Ausgang von Zahl einer 0 entspricht. Wenn die Münze fair ist, dann erscheinen 0 und 1 mit gleicher Wahrscheinlichkeit $1/2$. Wir erhalten genau die gleiche Information aus dem Auftreten eines der möglichen Ergebnisse. Die Entropie dieses Systems ist

$$H(S) = \sum_{i=1}^2 (1/2) \lg 2 = \lg 2 = 1$$

wenn wir uns entscheiden, binäre Logarithmen zu verwenden, wie wir es oft in der Informatik tun (eine andere Wahl von Logarithmen wird das Ergebnis nur um einen festen multiplikativen Faktor ändern).

Betrachten Sie nun stattdessen eine geladene Münze, bei der Kopf zweimal so oft wie Zahl erscheint. Somit tritt 1 mit Wahrscheinlichkeiten $2/3$ auf und 0 erscheint mit Wahrscheinlichkeiten $1/3$. Die Entropie dieses Systems ist

$$\begin{aligned}
 H(S) &= (2/3) \lg(3/2) + (1/3) \lg(3/1) \\
 &= (2/3) \lg(3/2) + (1/3) \lg 3 \\
 &= (2/3) \lg 3 - (2/3) \lg 2 + (1/3) \lg 3 \\
 &= \lg 3 - (2/3) \lg 2 \\
 &\approx 1.585 - (2/3) \\
 &\approx 0.918
 \end{aligned}$$

Wir erinnern uns an die intuitive Vorstellung von Entropie aus der Physik: ein Maß für die Zufälligkeit des Systems. Die Entropie des fairen Würfels ist größer als die des geladenen Würfels, weil die Ergebnisse des fairen Würfels zufälliger und weniger vorgeeignet sind als die Ergebnisse des geladenen Würfels.

2.2.4.4 Die Entropie des Englischen

Mit den Frequenzen aus Tab. 2.1 können wir die Entropie des Englischen berechnen. Mit den Cornell-Frequenzen und einem binären Logarithmus erhalten wir etwa 2,898 Bits Information pro Buchstabe. Wenn die Buchstabenhäufigkeit gleichmäßig wäre, wäre die Entropie einer 26-Buchstaben-Sprache etwa

$$H(S) = \lg 26 \approx 4.7$$

Bits pro Buchstabe. Somit sind etwa

$$100 \left(1 - \frac{2.898}{4.7} \right),$$

oder 38% der im Englischen verwendeten Buchstaben redundant.³

2.2.4.5 Entropie für gleichmäßige Verteilungen

Es lohnt sich als Benchmark darüber nachzudenken, was die Entropie für eine Verteilung ist, die im oben genannten Sinne eines fairen Würfels „fair“ ist. Betrachten Sie eine Menge von n Symbolen $S = \{a_1, \dots, a_n\}$, von denen jedes die gleiche Wahrscheinlichkeit $1/n$ hat. Für eine solche Verteilung ist die Entropie

³ Jahre vor Shannons Arbeit über Entropie veröffentlichte Mark Twain seinen humorvollen Beitrag „Ein Plan zur Verbesserung der Rechtschreibung in der englischen Sprache“, ein Teil davon war die Zusammenlegung von Buchstaben mit ähnlicher Funktion und Aussprache; dies hätte die Entropie erhöht, obwohl wir nicht wissen, dass jemals eine formale Berechnung durchgeführt wurde.

$$H(S) = \sum_{i=1}^n (1/n) \lg n = \lg n \sum_{i=1}^n (1/n) = \lg n.$$

Wenn man die Entropie als Maß für die Überraschung oder das Maß für die Menge an Informationen, die man erhält, wenn man a_k als nächstes Symbol in einer Sequenz sieht, betrachtet, ist dieser Wert das Maximum, das erreicht werden kann. Jede ungleiche Verteilung von Symbolen, wie die Häufigkeitszählungen von Buchstaben in jeder menschlichen Sprache, wird dazu führen, dass die Entropie kleiner ist als dieses Maximum.

2.3 Transpositionschiffren

Bei einer Substitutionschiffre werden die ursprünglichen Buchstaben durch verschiedene Buchstaben ersetzt. Bei einer Transposition Chiffre bleiben die Buchstaben gleich, aber ihre Reihenfolge wird geändert.

2.3.1 Säulentranspositionen

Wahrscheinlich die einfachste Form der Transposition besteht einfach darin, den Klartext in der normalen Reihenfolge über eine Seite zu schreiben, aber dann als Geheimtext die Buchstaben der Nachricht zu übertragen, die in Spalten nach unten gelesen werden. Zum Beispiel könnte „Eine einfache Nachricht wie dieser Satz“ so geschrieben werden

```
asimxpl
emessxa
gelxike
thissex
xtence
```

und dann könnte durch das Lesen der Spalten übertragen werden:

```
aegtn smehx ielit msxse xsise pxkec laexe
```

wo wir die Nachricht mit „x“ Zeichen aufgefüllt haben und dann Leerzeichen (die nicht Teil der übertragenen Nachricht wären) gezeigt haben, um sie in diesem Text lesbarer zu machen.

Variationen zu diesem Thema wurden seit Jahrtausenden verwendet. Die Spartaner des antiken Griechenlands verwendeten eine *Scytale*; indem man einen Papierstreifen um einen Stab mit einem festen Durchmesser wickelte, konnte man die Nachricht über den Stab schreiben und die einzelnen Buchstaben durch den Umfang des Stabes trennen. Ein Empfänger, der nicht im Besitz eines Stabes mit dem gleichen Durchmesser ist, würde die Buchstaben nicht so ausrichten können, wie sie ursprünglich geschrieben wurden.

2.3.2 Doppelte Transposition

Chiffren wie Playfair und ADFGX, die unten beschrieben werden, können leicht durch Statistiken über Buchstabendiagramme oder Buchstabenhäufigkeiten gebrochen werden. Aus diesem Grund wurden viele Transpositionschiffren als doppelte Transpositionen durchgeführt, wobei die erste die zugrunde liegenden Klartexte verschleierte und die zweite die für die Sicherheit benötigte Zufälligkeit vornahm.

2.4 Playfair

Angeblich wurde das erste Chiffrensystem, das zwei Buchstaben auf einmal verschlüsselte, von Sir Charles Wheatstone (der von der gleichnamigen Brücke) erfunden und nach seinem Freund Baron Playfair benannt, der ein großer Befürworter der Chiffre war. Man legt eine Tabelle mit Buchstaben aus, vielleicht

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

Beginnend mit einem Schlüsselwort (in diesem Fall der Name des Autors), lassen Sie Buchstaben aus, wenn sie wiederholt werden, beenden Sie mit Buchstaben, die nicht im Schlüsselwort enthalten sind, und verschmelzen Sie dann „i“ mit „j“, um ein quadratisches Tableau zu erzeugen.

Die Verschlüsselung erfolgt nun wie folgt. Beginnen Sie mit einer Nachricht

this is the message

und teilen Sie diese in Buchstabenpaare

th is is th em es sa ge

Drei Regeln bestimmen die Erzeugung des Geheimtextes.

1. Wenn die beiden Buchstaben des Paares in verschiedenen Zeilen und Spalten liegen, bilden sie die gegenüberliegenden Ecken eines Rechtecks, und das Geheimtextpaar sind die Buchstaben an den anderen beiden Ecken. Wir wählen den Eckbuchstaben, der in der gleichen Zeile wie der Klartextbuchstabe liegt.
2. Wenn die beiden Buchstaben des Paares in der gleichen Zeile liegen, verschieben Sie jeden Buchstaben um einen nach rechts.
3. Wenn die beiden Buchstaben des Paares in der gleichen Spalte liegen, verschieben Sie jeden Buchstaben um einen nach unten.

So wird th zu po, is zu mq, em zu fi, es zu fq, sa zu tc, ge zu bl, und der Geheimtext ist

po mq mq fi fq tc bl.

2.5 ADFGX

Ein Substitutions-Transpositions-Chiffre, das im Ersten Weltkrieg von den Deutschen intensiv genutzt wurde, wurde „ADFGX“ genannt. In seiner einfachsten Version handelt es sich einfach um eine Digramm-für-Digramm-Substitution. Gegeben ein Tableau

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

Mit einer zufälligen Auswahl für die 5×5 Matrix der Buchstaben, ersetzt man jeden Buchstaben durch das Zeilen-Spalten-Paar, an dem dieser Buchstabe gefunden wird. Unsere vorherige Nachricht,

this is the message

würde als der Geheimtext

GX FA FD GG FD GG GX FA DD FG DD GG GG AX DX DD

gesendet werden. Kompliziertere Versionen des Chiffres würden dann Transpositionen auf die Buchstaben des Substitutions-Geheimtextes anwenden.

2.6 Kryptoanalyse

2.6.1 Das Knacken eines Substitutions-Chiffres

Das Knacken eines Substitutions-Chiffres ist größtenteils eine Frage der Statistik, und mit Computerunterstützung müssen die mühsamen Teile nicht mehr von Hand erledigt werden. Eine Kombination aus Brute-Force, einer kleinen Menge an Baumsuche und Beschneidung unwahrscheinlicher Äste, und einigen Vermutungen und Krippen unter Verwendung von Häufigkeitszählungen von Buchstaben, Bigrammen und dergleichen, und ein einfacher Ersatz kann fast sofort geknackt werden.

Wir bemerken, dass Statistiken am besten funktionieren, wenn es Daten gibt, auf denen man Statistiken machen kann, und dass daher längere Nachrichten anfälliger für

Tab. 2.2 Häufigkeiten in der Gettysburg-Adresse

Lincoln	e	t	a	o	h	r	n	i	d
Cornell	e	t	a	o	i	n	s	r	h
Dickens	e	t	a	o	i	n	h	s	r
Darwin	e	t	a	o	n	i	s	r	h

statistische Angriffe sind als kurze Nachrichten. Allerdings müssen Nachrichten nicht so lang sein, um ihre Häufigkeitszählungen zu offenbaren.

Nehmen wir die Gettysburg-Adresse als Beispiel. Die neun häufigsten Buchstaben in Lincolns Adresse sind in Tab. 2.2 aufgeführt, zusammen mit den neun häufigsten Buchstaben, in der Reihenfolge, aus Tab. 2.1.

Die vier häufigsten Buchstaben stimmen überein und sie machen 42% der Buchstaben in der Adresse aus. Die nächsten fünf machen fast ein weiteres 32% der Gesamtbuchstaben aus. Wir würden erwarten, dass es relativ einfach wäre, die resultierenden Buchstabensequenzen gegen Englisch zu bewerten und den Baum auf etwas durchaus Machbares zu reduzieren, selbst wenn alle $10! = 3628800$ verschiedenen Permutationen ausprobiert wurden. Mit mehr als 70% der Buchstaben, die in den neun häufigsten Buchstaben enthalten sind, scheint es schwer zu glauben, dass ein vernünftiger Angriff scheitern würde.

Und wir weisen darauf hin, dass in dieser Argumentation die Fähigkeit, einen Computer zu benutzen, ein enormer Vorteil ist. Menschliche Kryptoanalytiker werden nicht drei Millionen Möglichkeiten ausprobieren, aber Computer können dies ganz leicht tun. Vor der Computerära und heute für diejenigen, die vielleicht das tägliche Zeitungskryptogramm von Hand machen, ist eine Menge gutes Rätselraten erforderlich. Mit Computern und Häufigkeitszählungen macht die zeitgeehrte Tradition von BFI⁴ macht einfache Substitution vollständig brechbar.

Wir weisen hier erneut darauf hin, dass das Trennen des Textes in Wörter einen Brute-Force-Angriff noch einfacher macht. Im gesamten Brown-Korpus aus dem Natural Language Tool Kit [4] gibt es nur 212 Zwei-Buchstaben- und 879 Drei-Buchstaben-Kombinationen, und viele davon treten als chemische Element- oder andere Abkürzungen auf.

2.6.2 Das Knacken eines Transpositionsschlüssels

Das Knacken eines Transpositionsschlüssels ist ebenfalls größtenteils eine Frage der Statistik und einiger Sprachvermutungen. Ein Beispiel für das Knacken eines solchen Schlüssels ist der Erste-Weltkrieg-Schlüssel in Anhang A.

⁴Brute Force und Ignoranz.

2.7 Das Vernam Einmal-Pad

Alle bis auf ein Kryptosystem, auch wenn man die modernen betrachtet, beruhen nicht auf der Fähigkeit, perfekt sicher zu sein, sondern auf der Annahme, dass es rechnerisch unpraktikabel ist, eine Nachricht zu entschlüsseln. Das allgemeine Ziel bei der Verschlüsselung einer Nachricht besteht darin, den Text zufällig erscheinen zu lassen. Es ist jedoch schwierig, völlig zufällig zu sein, daher verwenden sowohl klassische als auch moderne Kryptosysteme ein Muster oder eine Funktion, die den Text zufällig erscheinen lässt. Dieses Muster erleichtert die Verschlüsselung für Menschen, zeigt aber auch denen, die die Verschlüsselung angreifen, dass ein Muster existiert.

Das einzige wirklich sichere Kryptosystem ist ein einmaliges Pad. Eine Version eines Einmal-Pads wäre im Wesentlichen eine Vigenère-Chiffre, die das Alphabet auf der Grundlage einer unendlichen Folge von Zufallszahlen auswählt. Die Vigenère-Chiffre hat 26 Alphabete, aber aus Bequemlichkeit wird ein Schlüsselwort verwendet, um zu bestimmen, welches Alphabet für einen bestimmten Buchstaben verwendet wird. Wenn das Schlüsselwort aus Hunderttausenden von Wörterbuchwörtern, Ortsnamen, Personenamen usw. ausgewählt wird, ergibt sich eine scheinbare Zufälligkeit, aber es ist die endliche Länge des Schlüsselworts, die zu Wiederholungen bei der Auswahl der Alphabete und der Wahrscheinlichkeit führt, dass in einer langen Nachricht der gleiche Buchstabe mit dem gleichen Alphabet verschlüsselt wird. Wenn dieses „Schlüsselwort“ tatsächlich zufällig generiert und von unendlicher Länge wäre, würde die Wiederholung, die zur Erkennung der Länge führt, nicht im Geheimtext vorhanden sein.

Das Vernam Einmal-Pad [5] wurde etwa zur gleichen Zeit erfunden, als Teletypen zur Übertragung von Nachrichten verwendet wurden. Mit dem ursprünglichen Teletyp wurden Buchstaben in 5-Bit-Ganzzahlen umgewandelt und als solche übertragen, oft unter Verwendung von Lochstreifen. Eine Version eines Vernam Einmal-Pads hätte einen Begleitstreifen mit einer zufälligen Folge von 5-Bit-Ganzzahlen, und die Bits der zufälligen Ganzzahlen würden mit den Textbits XOR-iert, um den Geheimtext zu erzeugen. Der Empfänger, mit einem identischen Papierstreifen, würde erneut XOR-ieren, um den Klartext zu erzeugen.

Die Sicherheit der Vernam-Chiffre ist zu 100% garantiert, wie in [5] erwähnt, vorausgesetzt, dass

- es nur zwei Kopien des Schlüsselbands gibt;
- beide Seiten der Kommunikationsverbindung das gleiche Schlüsselband haben;
- das Schlüsselband nur einmal verwendet wird;
- das Schlüsselband wird sofort nach Gebrauch zerstört;
- das Schlüsselband enthält wirklich zufällige Zeichen;
- die Ausrüstung ist TEMPEST-sicher;
- das Schlüsselband wurde während des Transports nicht kompromittiert.

All diese Kriterien sind notwendig. Damit das Band wirklich zufällig ist, dürfen nur die beiden Kopien des Bandes existieren, die identisch sein müssen und nicht bei der Übertragung abgefangen werden dürfen. Um Wiederholungen zu vermeiden, wie sie zum Knacken einer Vigenère-Chiffre verwendet werden, muss das Band nur einmal verwendet und dann zerstört werden. Damit der Geheimtext gelesen werden kann, müssen die beiden Bänder natürlich identisch sein und synchronisiert starten, um den XOR-Prozess zu beginnen. TEMPEST-ing ist der Mechanismus, der verhindert, dass eine dritte Partei die von einem Gerät wie einem Teletyp oder Computerterminal erzeugten elektrischen Signale überwacht, und sowohl der Verschlüsselungs- als auch der Entschlüsselungsprozess müssen gegen „Mithören“ durch einen Gegner geschützt sein.

Und schließlich muss natürlich, um ein zufälliges XOR-ing zu sein, das nicht die Art von Wiederholung hat, die beim Knacken einer Vigenère-Chiffre verwendet wird, die Sequenz wirklich zufällig sein. Es gibt Mechanismen zur Erzeugung von Zufallszahlen. Es wurde behauptet, dass die britische Lotterie einmal einen Geigerzähler auf dem Dach ihres Gebäudes hatte, der das zufällige Muster von kosmischen Strahlen aufzeichnete, die auf den Zähler trafen. Viele Lotteriesysteme erzeugen wirklich zufällige Ergebnisse mit physischen Ping-Pong-Bällen, die sorgfältig auf Identität in Größe und Gewicht überprüft werden. Zufallszahlen wurden aus einer lauten Diode erzeugt.

Wir werden später die Erzeugung von Pseudo-Zufallszahlen mit einer periodischen Funktion diskutieren, deren Periode so lang ist, dass sie zufällig erscheint. Ein Großteil der Public-Key-Kryptographie beruht auf der Tatsache, dass Funktionen existieren, um Zahlen zu erzeugen, die Zufälligkeitstests bestehen, aber deterministisch mit einer Funktion erzeugt werden können.

2.8 Übungen

1. Folgendes wird als Chiffretext aus Caesar-Verschlüsselungen angenommen. Entschlüsseln Sie diese.
 - a. alirmrxligsyvwisjlygerizirxw
 - b. sdgkcdrolocdypdswocsdgkcdrogybcdypdswoc
 - c. hzruvadohafvbyjvbuayfjhukvmvyfjbhhrdohafvbjhukvmvyfjbvbyjvbuayf
2. (Programmierübung) Beschaffen Sie sich aus einer legitimen Quelle (vielleicht Project Gutenberg) Text in einer anderen Sprache als Englisch. (Französisch und Deutsch sind leicht von Project Gutenberg zu bekommen.) Berechnen Sie die Buchstabenhäufigkeiten für diese anderen Sprachen und vergleichen Sie sie mit Englisch, um zu bestimmen, wie Sie die zugrunde liegende Sprache eines Textes erkennen würden, der entweder mit einer Substitutions- oder Transpositionschiffre verschlüsselt ist.
3. (Programmierübung) Schreiben Sie ein Programm, das einem menschlichen Beobachter dabei hilft, einen Chiffretext aus einem englischen Dokument zu entschlüsseln, das mit einer Substitutionschiffre verschlüsselt wurde. Sie können davon ausgehen, dass die Wortgrenzen im Chiffretext vorhanden sind, so dass Sie etablierte

Listen von kurzen Wörtern als Krippen verwenden und längere Wörter basierend auf englischen Buchstabenmustern bewerten können.

4. (Programmierübung) Schreiben Sie ein Programm, um eine Chiffretextentschlüsselung aus einem englischen Dokument durchzuführen, das mit einer Substitutionschiffre verschlüsselt wurde. Im Gegensatz zur vorherigen Übung wird dies wahrscheinlich eine Tiefensuche durch mögliche Substitutionen und eine viel ausgefeiltere Bewertungsfunktion erfordern, um die Wahrscheinlichkeit zu bestimmen, dass die spezielle Zuweisung von Buchstaben an diesem Punkt im Suchbaum korrekt ist.
5. (Programmierübung) Schreiben Sie ein Programm, um nicht die einzelnen Buchstabenhäufigkeiten zu berechnen, sondern die Häufigkeiten von Zwei-Buchstaben- und Drei-Buchstaben-Sequenzen im Englischen. Sie könnten als Quelltext eines der Korpora aus dem Brown-Korpus oder einen Text aus dem Gutenberg-Projekt verwenden. Wenn Sie die Häufigkeitszählungen mit und ohne Wortgrenzen durchführen, sind sie dann unterschiedlich? Dies würde beeinflussen, wie Sie eine verschlüsselte Nachricht mit einem Suchprogramm angreifen würden.

2.8.1 Chiffretext für Substitutionschiffre-Probleme (3) und (4)

- mia zhjecad hg s mihjtscp kuqat baouct yumi s behfac gsc baqm scp qasfd muea
- ph chm ba ueeaxqsnsabqa ug dhj nscchm ba eaqsnap dhj nscchm ba xehkhmap
- ug dhj miucf chbhpd nseat ug dhj sea squva med kuttuco s nhjxqa hg nse xsdkacmt
- baghea dhj neumunuwa thkahca dhj tihjqp ysqr s kuqa uc miaue tihat mism ysd yiac dhj neumunuwa miak dhj sea s kuqa sysd scp dhj isva miaue tihat
- ug sm guetm dhj ph chm tjnnaap tfdpuvuco ut chm ghe dhj
- ug dhj maqq mia mejmi dhj ph chm isva mh eakakbae scdmiuco
- mia ljunfatm ysd mh phjbqa dhje khcad ut mh ghqp um uc isgg scp xjm um bsnf uc dhje xhnfam
- pjnm msxa ut qufa mia ghena um ist s quoim tupa scp s psef tupa scp um ihqpt mia jcuvaeta mhoamiaie
- oacaesqqd txasfuo dhj sea chm qasecuco kjni yiac dhje quxt sea khvuco
- arxaeuacna ut thkamiuco dhj phc m oam jcmuq zjtm sgmae dhj caap um
- cavae kutt s ohhp niscna mh tijm jx
- ouva s ksc s guti scp ia yuqq asm ghe s psd masni iuk ihy mh guti scp ia yuqq tum uc s bhsn scp peucf baee sqq psd

Literatur

1. Cornell University, English letter frequency, <http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html>
2. Project Gutenberg, The Gutenberg Project, <https://www.gutenberg.org>

3. R. W. Hamming, *Coding and Information Theory*, 2. Aufl. (Prentice-Hall, Hoboken, 1986), S. 103ff
4. Natural Language Tool Kit, Brown Corpus, <http://www.nltk.org>
5. The Crypto Museum, The Vernam cipher, <https://www.cryptomuseum.com/crypto/vernam.htm>.
Zugegriffen 24 Januar 2020



Teilbarkeit, Kongruenzen und modulare Arithmetik

3

Zusammenfassung

Die moderne Kryptographie basiert weitgehend auf den Mathematiken der modularen Arithmetik, Kongruenzen und der Arithmetik in den ganzen Zahlen modulo Primzahlen oder Produkte von (meistens) zwei großen Primzahlen. In diesem Kapitel behandeln wir die grundlegende Zahlentheorie, die in symmetrischen und asymmetrischen kryptographischen Systemen vorkommt: Teilbarkeit und Kongruenzen, größter gemeinsamer Teiler, Exponentiation und die Euler'sche Totient. Unser Schwerpunkt liegt auf mathematischen Theoremen, die verstanden und angewendet werden müssen, anstatt auf ihren Beweisen, es sei denn, die Methode oder Konstruktionen in den Beweisen sind relevant für die Kryptographie selbst. Obwohl wir dies als Hintergrundmathematik behandeln, weisen wir darauf hin, dass der Leser leicht Beispiele für alle abgedeckten Prinzipien generieren kann sowie Beispiele finden kann, die demonstrieren, warum die gemachten Annahmen notwendig sind und die Schlussfolgerungen eng gezogen sind.

3.1 Teilbarkeit

Wir gehen in diesem Kapitel von der Annahme aus, die nicht immer ausdrücklich erwähnt wird, dass alles eine ganze Zahl ist.

Definition 3.1 Wir sagen, dass eine ganze Zahl a eine ganze Zahl b *teilt*, und schreiben alb , wenn es eine dritte ganze Zahl d gibt, so dass $b = ad$. Wir nennen ein solches a einen *Teiler* oder *Faktor* von b . Wenn a ein Teiler der ganzen Zahlen b und c ist, dann sagen wir, dass a ein *gemeinsamer Teiler* von b und c ist. Wenn wir alc und $b|c$ haben, dann nennen wir c ein *gemeinsames Vielfaches* von a und b .

Satz 3.1

1. Für alle c , alb impliziert, dass $albc$.
2. Wenn alb und $b|c$, dann alc .
3. Für alle ganzen Zahlen x und y , wenn alb und alc , dann $a|(bx + cy)$.
4. Wenn alb und $b|a$, dann $a = \pm b$.
5. Wenn alb und beide $a > 0$ und $b > 0$, dann $a \leq b$.

Beweis

1. Wenn alb , dann gibt es ein d , so dass $b = ad$. Für alle c haben wir dann $bc = adc$, und dc ist der Multiplikator, der die Bedingung in der Definition von „teilt“ erfüllt.
2. Wenn $b = ad_1$ und $c = bd_2$ für Ganzzahlen d_1, d_2 , dann $c = ad_1d_2 = a(d_1d_2)$ und wir haben erneut die Bedingung für „teilt“ erfüllt.
3. Wenn $b = ad_1$ und $c = ad_2$ für Ganzzahlen d_1, d_2 , dann für alle $x, y \in \mathbb{Z}$ haben wir

$$bx + cy = ad_1x + ad_2y = a(d_1x + d_2y)$$

und erneut haben wir die Bedingung erfüllt.

4. Wenn $b = ad_1$ und $a = bd_2$ für Ganzzahlen d_1, d_2 , dann haben wir $b = ad_1 = bd_1d_2$ und somit $d_1d_2 = 1$. Dies kann nur gelten, wenn $d_1 = d_2 = +1$ oder $d_1 = d_2 = -1$.
5. Wenn $b = ad$, und diese sind positiv, dann sind die einzigen möglichen Werte für b $a, 2a, 3a, \dots$, alle größer als a außer dem ersten, der gleich ist.

Definition 3.2 (*Der Division Algorithmus*) Gegeben zwei Elemente $a, b \in \mathbb{Z}$, der *Division Algorithmus* ist der Prozess, durch den wir die Ganzzahlen q und r , den *Quotienten* und *Rest*, mit $0 \leq r < a$, so dass

$$b = qa + r$$

1. Unter den endlich vielen Werten $r = b - qa$, für $|q| \leq |b|$, wählen wir den nicht-negativen Wert mit der geringsten Größe.
2. Geben Sie r und $q = (b - r)/a$ zurück.

Satz 3.2 Gegeben Ganzzahlen a und $b \neq 0$, der Division Algorithmus liefert Werte q und r so dass $b = qa + r$ und $0 \leq r < |a|$.

Satz 3.3 Es gibt nur endlich viele Teiler einer beliebigen Ganzzahl a .

Definition 3.3 Für a und b Ganzzahlen, nicht beide null, wird der größte gemeinsame Teiler von a und b als *größter gemeinsamer Teiler*, geschrieben $\gcd(a, b)$, bezeichnet. Das kleinste gemeinsame Vielfache von a und b wird als *kleinstes gemeinsames Vielfaches*, geschrieben $\text{lcm}(a, b)$, bezeichnet.

Bemerkung 3.1 Der ggT wird häufig nur als (a, b) und das kgV häufig nur als $[a, b]$ geschrieben. Es gibt jedoch zu viele Dinge ohne ausreichende Kennzeichnung, daher werden wir sie nicht auf diese Weise schreiben, sondern so wie in der Definition.

Satz 3.4 Gegeben a und b , nicht beide null, sind die Werte von $\gcd(a, b)$ und $\text{lcm}(a, b)$ einzigartig.

Satz 3.5 Die $\gcd(a, b)$ ist der kleinste positive Wert von $ax + by$, während x und y durch alle Ganzzahlen laufen.

Beweis Dies ist ein nützlicher konstruktiver Beweis, also werden wir ihn durchführen.

Zunächst lassen Sie $g = \gcd(a, b)$. Dann haben wir $g|(ax + by)$ für alle x und y nach Theorem 3.1, Teil 3.

Lassen wir $\ell = ax_0 + by_0$ der kleinste positive Wert von $ax + by$ sein, während x und y alle Ganzzahlen durchlaufen. Da g alle solche Werte teilt, haben wir, dass $g|\ell$. Da sowohl g als auch ℓ positiv sind, wissen wir, dass es eine Ganzzahl $k \geq 1$ gibt, so dass $\ell = gk$. Aber wenn $k > 1$, dann ist g streng kleiner als ℓ , was im Widerspruch zu unserer Annahme steht. Daher muss es so sein, dass $k = 1$ und $g = \ell$. \square

Bemerkung 3.2 Beachten Sie, dass der vorhergehende Satz besagt, dass wenn $g = \gcd(a, b)$, dann können wir x_0 und y_0 finden, so dass $g = ax_0 + by_0$. Das ist eine sehr große Sache.

Bemerkung 3.3 Beachten Sie, dass wenn a b teilt, dann $\gcd(a, b) = |a|$ und ist nicht null, weil wir den ggT als positiv und nicht nichtnegativ definiert haben.

Satz 3.6 Nehmen Sie an, dass a und b beide nicht null sind.

1. Wenn $g = \gcd(a, b)$, dann teilt g jeden gemeinsamen Teiler von a und b .
2. Wenn $m > 0$, dann $\gcd(ma, mb) = m \cdot \gcd(a, b)$.
3. Wenn $m > 0$, dann $\text{lcm}(ma, mb) = m \cdot \text{lcm}(a, b)$.
4. Wenn $d > 0$, $d|a$, und $d|b$, dann $\gcd(a, b) = d \gcd(a/d, b/d)$.
5. Wenn $g = \gcd(a, b)$, dann $\gcd(a/g, b/g) = 1$.
6. $\gcd(a, b) = \gcd(b, a)$.
7. $\text{lcm}(a, b) = \text{lcm}(b, a)$.
8. $\gcd(a, -b) = \gcd(a, b)$.
9. $\text{lcm}(a, -b) = \text{lcm}(a, b)$.
10. $\gcd(a, b + xa) = \gcd(a, b)$ für alle Ganzzahlen x .
11. $\gcd(a, 0) = \gcd(0, a) = \gcd(a, a) = \text{lcm}(a, a) = |a|$.
12. $\gcd(a, b, c) = \gcd(a, \gcd(b, c))$.
13. Wenn $clab$, und wenn $\gcd(b, c) = 1$, dann cla .

Definition 3.4 Wenn $\gcd(a, b) = 1$, dann sagen wir, dass a und b *relativ prim* oder *zueinander prim* sind.

3.2 Der euklidische Algorithmus

Der euklidische Algorithmus ist vielleicht der älteste Algorithmus auf dem Planeten. Sicherlich ist er wahrscheinlich der älteste Algorithmus, der noch in seiner ursprünglichen Form verwendet wird. Die naive Version hier wird fast genau so präsentiert wie in Euklids *Elementen*.

3.2.1 Der naive euklidische Algorithmus

Algorithm 3.1 Naive algorithm to calculate $g = \gcd(a, b)$

Require: $a, b \in \mathbb{Z}$, not both zero

1: $r_{-1} \leftarrow a$

2: $r_0 \leftarrow b$

3: $j \leftarrow 0$

4: **while** $r_j \neq 0$ **do**

5: $j \leftarrow j + 1$

6: Use the division algorithm to obtain q_j and r_j : $r_{j-2} \leftarrow q_j r_{j-1} + r_j$

7: **end while**

8: Output $g \leftarrow r_{j-1}$

Beispiel 3.1 Berechnen wir den ggT von 366 und 252.

$$\begin{aligned}
 r_{-1} &= 366 \\
 r_0 &= 252 \\
 j &= 0 \\
 j &= 1 \\
 q_1 &= 1 \quad r_1 = 114 \quad 366 = 1 \cdot 252 + 114 \\
 j &= 2 \\
 q_2 &= 2 \quad r_2 = 24 \quad 252 = 2 \cdot 114 + 24 \\
 j &= 3 \\
 q_3 &= 4 \quad r_3 = 18 \quad 114 = 4 \cdot 24 + 18 \\
 j &= 4 \\
 q_4 &= 1 \quad r_4 = 6 \quad 24 = 1 \cdot 18 + 6 \\
 j &= 5 \\
 q_5 &= 3 \quad r_5 = 0 \quad 18 = 3 \cdot 6 + 0
 \end{aligned}$$

Wir geben $r_4 = 6$ als den ggT aus.

Bemerkung 3.4 Wir stellen fest, dass der Algorithmus enden muss, weil der Divisionalgorithmus bei jedem Schritt einen kleineren, nichtnegativen Wert von r_j erzeugt, so dass der Prozess nicht ewig weitergehen kann.

Bemerkung 3.5 Wir stellen fest, dass eine schlechteste Laufzeit dieser Version des euklidischen Algorithmus auftritt, wenn a und b aufeinanderfolgende Fibonacci-Zahlen sind, weil in diesem Fall alle Quotienten 1 sind und die Anzahl der Schritte maximiert ist. Da es eine geschlossene Form für die Fibonacci-Zahlen gibt,

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

und die ggT-Operation n Schritte hinabsteigt, bis wir zum Basiswert 1 gelangen, der der ggT von zwei benachbarten Fibonacci-Zahlen ist, wissen wir, dass die Anzahl der Schritte, die in diesem schlechtesten Fall, n , zu nehmen sind, logarithmisch in der Größe von F_n ist. Tatsächlich, in dem Wissen, dass wir $\gcd(a, b) = \gcd(b, a - b)$ schreiben können, wenn wir annehmen, dass $a > b$ zu Beginn und dass alle Quotienten 1 in der Abstieg zum ggT sind, haben wir

$$\begin{aligned}\gcd(a, b) &= \gcd(b, a - b) \\ &= \gcd(a - b, 2b - a) \\ &= \gcd(2b - a, 2a - 3b) \\ &= \gcd(2a - 3b, 5b - 3a)\end{aligned}$$

und so weiter. Die Koeffizienten sind die Fibonacci-Zahlen, mit wechselnden Vorzeichen, und die logarithmische Anzahl der Schritte ist unabhängig von den Werten von a und b .

3.2.2 Der erweiterte euklidische Algorithmus

Unser Beispiel oben verdient eine explizitere Erläuterung. Der naive euklidische Algorithmus findet den größten gemeinsamen Teiler g der Ganzzahlen a und b . Der erweiterte Algorithmus findet Werte x und y so, dass $ax + by = g$, und es erfordert nur, dass wir die notwendigen Koeffizienten im Auge behalten. Wenn wir in Folge rechnen, mit dem Division Algorithmus,

$$\begin{aligned}a &= r_{-1} \\ b &= r_0 \\ r_{-1} &= r_0 q_1 + r_1 \\ r_0 &= r_1 q_2 + r_2 \\ r_1 &= r_2 q_3 + r_3 \\ r_2 &= r_3 q_4 + r_4\end{aligned}$$

Dies ermöglicht es uns, nachzuvollziehen, wie man r_i aus den ursprünglichen Werten von a und b berechnet:

$$\begin{aligned} r_1 &= a - q_1 b \\ r_2 &= b - q_2 r_1 = b - q_2(a - q_1 b) = (q_1 q_2 + 1)b - q_2 a \\ r_3 &= r_1 - q_3 r_2 = a - q_1 b - q_3((q_1 q_2 + 1)b - q_2 a) \\ &= (q_2 q_3 + 1)a - (q_1 q_2 q_3 + q_3 + q_1 + 1)b \end{aligned}$$

und so weiter. Die Algebra wird hässlich, aber rekursiv als Computerprogramm durchgeführt, ist jeder Schritt einfach. Jeder der Reste im euklidischen Algorithmus kann als eine ganzzahlige lineare Kombination der ursprünglichen a und b ausgedrückt werden. Für unser Beispiel oben berechnen wir

$$\begin{aligned} 114 &= 366 - 252 \\ 24 &= 3 \cdot 252 - 2 \cdot 366 \\ 18 &= 9 \cdot 366 - 13 \cdot 252 \\ 6 &= 16 \cdot 252 - 11 \cdot 366 \end{aligned}$$

Dies führt uns zu einer enorm wichtigen Tatsache.

Satz 3.7 Lassen Sie x_0 und y_0 jedes Ergebnis sein, das aus dem erweiterten euklidischen Algorithmus zur Bestimmung

$$ax_0 + by_0 = \gcd(a, b)$$

Dann ist die Menge aller Lösungen von $ax + by = \gcd(a, b)$ genau die Menge

$$(x, y) = (x_0 + \lambda b, y_0 - \lambda a)$$

für $\lambda \in \mathbb{Z}$.

3.2.3 Der Binäre Euklidische Algorithmus

Die naive Version von Euklid erfordert eine Division, die auf einem Computer die mit Abstand langsamste aller arithmetischen Operationen ist. Schon in alten Zeiten,¹ lag das relative Kostenverhältnis von Addition gegenüber Multiplikation gegenüber Divisionin

¹ Die 1970er Jahre?

der Größenordnung von eins zu fünf oder eins zu zehn.² Auf modernen Computern kann seit der Einführung von RISC-Architekturen in den 1980er Jahren eine Ganzzahldivision bis zu 100 einzelne Maschinenbefehle erfordern. Diese Kosten werden durch die langen Pipelines auf modernen CPUs etwas gemildert, aber die Kosten der Ganzzahldivision sind immer noch sehr hoch im Vergleich zu Addition oder Multiplikation. Dies wird in gewissem Maße noch verschlimmert durch die Tatsache, dass viel Aufmerksamkeit auf die Fließkommaarithmetik gelegt wird, in der Hoffnung, die Leistung der wissenschaftlichen Datenverarbeitung zu verbessern, aber wenig Aufmerksamkeit wird der Ganzzahlarithmetik geschenkt, für die die Kryptographie fast die einzige wirklich rechenintensive Anwendung ist, die Ganzzahlarithmetik für etwas anderes als die Steuerung von Schleifen oder das Indizieren in Speicherarrays verwendet.

Wir beobachten drei Dinge:

1. Wenn a und b beide ungerade sind, dann $\gcd(2^i a, 2^j b) = 2^{\min(i,j)} \gcd(a, b)$;
2. Wenn a und b beide ungerade sind, dann ist $b - a$ gerade;
3. $\gcd(a, b) = \gcd(a, b - a)$.

Unser binärer ggT-Algorithmus ist daher der folgende.

Algorithm 3.2 Binary algorithm to calculate $g = \gcd(a, b)$

Require: $a, b \in \mathbb{Z}$, not both zero

- 1: $a_s \leftarrow a$ right shifted i bits until the rightmost bit of a_s is a 1
 - 2: $b_s \leftarrow b$ right shifted j bits until the rightmost bit of b_s is a 1
 - 3: Comment: Note that $g = 2^{\min(i,j)} \gcd(a_s, b_s)$
 - 4: **while** a_s and b_s are both nonzero **do**
 - 5: **if** $a_s > b_s$ **then**
 - 6: exchange a_s and b_s
 - 7: **end if**
 - 8: $b_s \leftarrow b_s - a_s$
 - 9: $b_s \leftarrow b_s$ right shifted k bits until the rightmost bit of b_s is a 1
 - 10: **end while**
 - 11: Output $g \leftarrow 2^{\min(i,j)} a_s$
-

²Auf dem IBM System 370 Model 158 der frühen 1970er Jahre zum Beispiel kostete die Ganzzahlmultiplikation 6,5-mal so viel wie die Ganzzahladdition, und die Ganzzahldivision kostete 47-mal so viel, wobei die Division also etwa 7-mal so teuer war wie die Multiplikation.

Der Algorithmus, kurz gesagt, ist folgender. Wir wissen, dass der ggT von a und b die kleinere Potenz von 2 ist, die a und b teilt, multipliziert mit dem ggT der ungeraden Teile von a und b . Wir entfernen also zuerst die Potenzen von 2, die in der Binärdarstellung die rechten Nullbits sind. Dann subtrahieren wir die kleinere von der größeren Zahl. Da die Differenz zweier ungerader Zahlen gerade ist, wissen wir, dass wir mindestens ein Nullbit von der Differenz verschieben können und wiederholen dies mit dem nun kleineren Paar von Ganzzahlen. Wenn wir subtrahieren, um Null zu erhalten, dann ist der positive Wert von a , der ungerade Teil des ggT.

Was bedeutend ist, ist, dass dieser Algorithmus keine Multiplikation und keine Division erfordert. Er verwendet nur die Addition, Bit-Test und Subtraktionsbefehle, die zu den schnellsten Befehlen auf jedem Computer gehören.

Der Nachteil dieses Algorithmus bei 64-Bit-Ganzzahlen ist, dass wir die Schleife viele Male durchlaufen müssten; die pathologisch schlechtesten Fälle wären diejenigen, bei denen wir bei jeder Subtraktion nur ein Nullbit von der Differenz verschieben können, und wenn ein Operand viel kleiner ist als der andere, sieht das Subtrahieren und Verschieben eher aus wie das, was die Hardware viel effizienter machen würde. Das Potenzial für so viele Schleifendurchläufe müsste gegen die relativen Kosten der Maschinenbefehle, die innerhalb der Schleife verwendet werden, abgewogen werden.

3.2.4 Der Dreimal-Subtrahieren-Euklidische Algorithmus

Paul Lévy [1] zeigte, dass für zufällige Ganzzahlen a und b derselben maximalen Bitlänge das Quotient a/b etwa 41,5 % der Zeit 1, etwa 17,0 % der Zeit 2 und etwa 9,3 % der Zeit 3 war, und somit war es 1, 2 oder 3 etwas mehr als zwei Drittel der Zeit. Dies führt zu einem Hybridalgorithmus, bei dem man die kleinere von a oder b so lange subtrahiert, bis das Ergebnis negativ ist, oder bis zu viermal. In etwa 2/3 der Fälle kann dann die Division vermieden werden.

3.2.5 GGT von großen Ganzzahlen

Algorithm 3.3 Subtraction algorithm to calculate $g = \gcd(\textit{bigger}, \textit{smaller})$

Require: $\textit{bigger}, \textit{smaller} \in \mathbb{Z}$, not both zero, with $\textit{bigger} > \textit{smaller}$ **while** $\textit{smaller} > 0$ **do** $\textit{bigger} \leftarrow \textit{bigger} - \textit{smaller}$ **if** $\textit{bigger} < \textit{smaller}$ **then** $q \leftarrow 1$ Exchange \textit{bigger} and $\textit{smaller}$ **else** $\textit{bigger} \leftarrow \textit{bigger} - \textit{smaller}$ **if** $\textit{bigger} < \textit{smaller}$ **then** $q \leftarrow 2$ Exchange \textit{bigger} and $\textit{smaller}$ **else** $\textit{bigger} \leftarrow \textit{bigger} - \textit{smaller}$ **if** $\textit{bigger} < \textit{smaller}$ **then** $q \leftarrow 3$ Exchange \textit{bigger} and $\textit{smaller}$ **else**Compute q and r using the division algorithm**end if****end if****end if**Exchange \textit{bigger} and $\textit{smaller}$ if necessary so $\textit{bigger} > \textit{smaller}$ **end while**Output $g \leftarrow \textit{bigger}$

Die Public-Key-Kryptographie, über die wir später in diesem Buch sprechen werden, basiert auf Arithmetik modulo großen Ganzzahlen, vielleicht von 1024, 2048 oder sogar 4096 Bits Länge. Wie man sich vorstellen kann, könnten gute Algorithmen zur Berechnung des größten gemeinsamen Teilers für solche großen Ganzzahlen anders sein als naivere Algorithmen. Die Grundversion des Euklidischen Algorithmus erfordert Division, die bei langen Ganzzahlen sehr teuer sein kann. Der binäre Algorithmus kann nur garantieren, dass die Größe der Ganzzahlen um ein Bit auf einmal reduziert wird. Die Division selbst unterscheidet sich von Addition, Subtraktion und Multiplikation, da die Schülerarithmetik für die Division von den signifikantesten bis zu den am wenigsten signifikanten Ziffern arbeitet, nicht umgekehrt, und im Gegensatz zu den anderen drei Operationen kann es bei der Division passieren, dass die Testquotientenziffer zu groß ist. Teilt man beispielsweise 30 durch 60, so ist der Testquotient von $6/3 = 2$ korrekt, aber teilt man 31 durch 60, so ergibt sich der korrekte Quotient von 1 erst, wenn man die ersten beiden Ziffern betrachtet. Man kann leicht zeigen, dass für Ganzzahlen mit vielen Ziffern keine festgelegte Gruppe von Testziffern immer den korrekten Quotienten liefert. Eine verbesserte Version des Euklidischen Algorithmus, geeignet für GGT von sehr langen Ganzzahlen, aber nur mit festen Einfachpräzisionsarithmetik, wurde von Lehmer [2] gegeben und wird auch in Knuth [1, S. 328ff] vorgestellt und analysiert. Lehmers Verbesserung des Euklidischen Algorithmus liefert einen vorhergesagten Quotienten, der fast immer korrekt ist, aber mit der Einfachpräzisionsarithmetik der führenden Ziffern von Divisor und Dividend berechnet werden kann.

Lehmers Version des Euklidischen Algorithmus arbeitet von den signifikantesten bis zu den am wenigsten signifikanten Ziffern. Ein alternativer Algorithmus zur Berechnung des GGT stammt von den Arbeiten von Sorenson und dann Jebelean, Weber und Sedjelmaci [3–6]. Dieser Algorithmus arbeitet von niedrigen zu hohen Ziffern und löst

$$au + bv \equiv 0 \pmod{k}$$

für Werte a, b , die durch \sqrt{k} begrenzt sind. Die Verwendung von Werten für k , die Potenzen von 2 sind, ermöglicht die Berechnung der Reduktion modulo k durch Extrahieren von Bits; wir werden diesen Ansatz ausführlicher in Kap. 8 sehen.

3.3 Primzahlen

Definition 3.5 Eine ganze Zahl p wird als *prim* bezeichnet, wenn die einzigen Teiler von p 1 und p sind. Eine ganze Zahl $p \neq 1$, die nicht prim ist, wird als *zusammengesetzt* bezeichnet.

Bemerkung 3.6 Wir gehen davon aus, dass wir den Begriff „prim“ nur auf positive ganze Zahlen anwenden.

Bemerkung 3.7 Wir stellen fest, dass nach moderner Konvention 1 nicht als Primzahl gilt. Dies war nicht immer die Konvention. D. N. Lehmer zählte 1 als Primzahl.

Satz 3.8 (Hauptsatz der Arithmetik) Die Zerlegung einer ganzen Zahl n in ein Produkt von Primzahlen ist eindeutig bis auf die Reihenfolge der Primzahlen und die Multiplikation mit $+1$ oder -1 .

Satz 3.9 (Euklid) Die Anzahl der Primzahlen ist unendlich.

Beweis (Dies ist genau der Beweis, der in Euklids Buch IX, Proposition 20, gegeben wird.) Wir führen einen Beweis durch Widerspruch. Nehmen wir an, die Anzahl der Primzahlen ist endlich, und lassen Sie P die Menge aller Primzahlen sein. Betrachten Sie die ganze Zahl $N = \left(\prod_{p \in P} p\right) + 1$. Jetzt kann dies aufgrund des Theorems 3.8 als Produkt von Primzahlen geschrieben werden, und somit gibt es eine Primzahl p , die N teilt. Allerdings, $p \notin P$, denn dann hätten wir $p|N$ und $p|\prod_{p \in P} p$ und somit $p|1$, was unmöglich ist. Also ist diese Primzahl p eine Primzahl, die nicht in P enthalten ist; unsere vermeintliche Menge P aller Primzahlen ist nicht eine Menge aller Primzahlen, und das ist ein Widerspruch. Unsere Annahme, dass die Anzahl der Primzahlen endlich ist, muss falsch sein. \square

Schließlich haben wir ein Theorem, das zeigt, dass die Primzahlen tatsächlich die Bausteine der Teilbarkeit der ganzen Zahlen sind.

Satz 3.10 Wenn p eine Primzahl ist und wenn $plab$, dann gilt entweder pla oder plb .

3.4 Kongruenzen

Definition 3.6 Zwei Ganzzahlen a und b werden als *kongruent modulo* einer positiven Ganzzahl m bezeichnet, wenn $m|(a - b)$. Wir schreiben $a \equiv b \pmod{m}$ und sagen, dass a *kongruent zu b modulo m* ist. Die Ganzzahl m wird als *Modulus* bezeichnet.

Definition 3.7 Wenn $a \equiv b \pmod{m}$ und wenn wir haben $0 \leq a < b$, sagen wir, dass a das *kleinste positive Residuum modulo m* ist.

Definition 3.8 Eine *vollständige Menge von Residuen modulo m* ist eine Menge S , so dass jede Ganzzahl n kongruent zu einem Element von S modulo m ist, aber keine zwei Elemente von S zueinander modulo m kongruent sind.

Wir nehmen normalerweise als unsere vollständige Menge von Residuen modulo m entweder die Menge der kleinsten positiven Residuen

$$S = \{r : 0 \leq r < m\}$$

oder (gelegentlich, wenn es uns passt) die Menge der Residuen mit der kleinsten Betragsgröße

$$S' = \{r : -m/2 < r \leq m/2\}.$$

Die nächste Reihe von Aussagen und Theoremen formuliert in konkreten Begriffen Ergebnisse, die wir als wahr annehmen würden, wenn wir zuerst den Stoff von Kap. 4 durchgearbeitet und bewiesen hätten, dass die Ganzzahlen modulo m einen Ring bilden.

Satz 3.11 Kongruenz modulo m ist eine Äquivalenzrelation.

1. (Reflexivität) $a \equiv a \pmod{m}$;
2. (Symmetrie) $a \equiv b \pmod{m}$ wenn und nur wenn $b \equiv a \pmod{m}$;
3. (Transitivität) $a \equiv b \pmod{m}$ und $b \equiv c \pmod{m}$ impliziert, dass $a \equiv c \pmod{m}$.

Satz 3.12

1. $a \equiv b \pmod{m}$ und $c \equiv d \pmod{m}$; impliziert, dass für alle Ganzzahlen x, y wir haben $ax + cy \equiv bx + dy \pmod{m}$;
2. $a \equiv b \pmod{m}$ und $c \equiv d \pmod{m}$ impliziert, dass $ac \equiv bd \pmod{m}$;
3. $a \equiv b \pmod{m}$ impliziert, dass für jedes Polynom $f(x)$ mit ganzzahligen Koeffizienten wir haben $f(a) \equiv f(b) \pmod{m}$.

Die Aussagen im vorherigen Satz handeln davon, was passiert, wenn wir multiplizieren, und wir wissen, dass Multiplikation Sinn ergibt.

Im Allgemeinen sind wir jedoch nicht in dem, was wir in Kap. 4 als ein Feld lernen werden, und die Division ist nicht garantiert mit dem ursprünglichen Modulus möglich; die Division in Kongruenzen beinhaltet den ggT des Modulus und des Dividenden.

Satz 3.13 Folgendes ist wahr.

1. $ax \equiv ay \pmod{m}$ wenn und nur wenn $x \equiv y \pmod{m/(\gcd(a, m))}$;
2. $ax \equiv ay \pmod{m}$ und $\gcd(a, m) = 1$ impliziert, dass $x \equiv y \pmod{m}$;
3. Die Lösung einer linearen Kongruenz

$$ax \equiv b \pmod{m}$$

ist gleichwertig mit der Durchführung des erweiterten euklidischen Algorithmus. Wenn wir

$$\gcd(a, m) \nmid b$$

haben, dann existiert keine solche Lösung.

Beweis Wir werden nur das Letzte davon beweisen. Nehmen wir an, wir haben eine lineare Kongruenz

$$ax \equiv b \pmod{m}$$

die für die Unbekannte x gelöst werden soll. Dies entspricht der Bestimmung von x so, dass

$$ax - b = my$$

für eine bestimmte Unbekannte y , und dies entspricht der Suche nach x und y so, dass

$$ax - my = b.$$

Offensichtlich, wenn $\gcd(a, m) \nmid b$ dann kann die Gleichung nicht gelöst werden, weil der ggT die linke Seite teilt, aber nicht die rechte Seite.

Wenn $\gcd(a, m) \mid b$ dann können wir den erweiterten euklidischen Algorithmus durchführen, um x_0 und y_0 so zu berechnen, dass

$$ax_0 - my_0 = g,$$

wo $g = \gcd(a, m)$. Dies entspricht der Lösung der Gleichung

$$(a/g)x_0 - (m/g)y_0 = 1.$$

Wir können nun zurückmultiplizieren, um

$$a(b/g)x_0 - m(b/g)y_0 = b,$$

zu erhalten, wo $a, b/g, x_0, m, b/g, y_0$ alle ganze Zahlen sind, und wir haben eine integrale Lösung für die notwendige Gleichung.

Die Tatsache, dass der erweiterte euklidische Algorithmus alle Lösungen für die lineare Kongruenz liefert, stammt aus Theorem 3.7. \square

Bemerkung 3.8 Es sollte beachtet werden, dass der Begriff *modulare Arithmetik* und nicht *Modulo-Arithmetik* lautet. Ersterer wird seit etwa den 1830er Jahren in Englisch verwendet, und es ist bedauerlich, dass der falsche Begriff anscheinend in den allgemeinen Gebrauch übergegangen ist.

Bemerkung 3.9 Wir werden die kleinste Restlösung von

$$ax \equiv 1 \pmod{m}$$

als $a^{-1} \pmod{m}$ bezeichnen, so dass wir die Multiplikation durch ein Inverses modulo m , wenn das gut definiert ist, als Ganzzahlarithmetik modulo m darstellen können, und wir werden weiterhin so etwas wie $1/a$ schreiben, wenn die „Division“ möglicherweise keine Ganzzahl ergibt.

Satz 3.14 (Chinesischer Restsatz (CRT)) Lassen k positive Ganzzahlen sein, die paarweise teilerfremd sind (das heißt, für die $\gcd(m_i, m_j) = 1$ für jede $i \neq j$ gilt). Dann gibt es für jede Menge von Ganzzahlen a_1, a_2, \dots, a_k , die simultanen Kongruenzen

$$x \equiv a_i \pmod{m_i}$$

eine eindeutige Lösung X modulo $M = \prod_{i=1}^k m_i$.

Beweis (Erste Version) Der erste Beweis ist eine Variation der Lagrange-Interpolation und wird durchgeführt, indem man den Hasen in einem Schritt aus dem Hut zieht.

Lassen Sie $M_j = M/m_j$. Da die m_i paarweise teilerfremd sind, wissen wir, dass $\gcd(M_j, m_j) = 1$ und somit können wir b_j finden, so dass $M_j b_j \equiv 1 \pmod{m_j}$. (Wir behaupteten, dass Theorem 3.5 eine große Sache war. Wir haben es gerade hier verwendet.)

Wir lassen dann

$$X = \sum_{i=1}^k a_i b_i M_i.$$

Nun, für jede m_j , haben wir, dass $m_j | M_i$ für $i \neq j$, und somit dass

$$X \equiv a_j b_j M_j \equiv a_j \cdot 1 \pmod{m_j}.$$

Beweis (Zweite Version) Die erste Kongruenz ist einfach: die Lösungen von

$$x \equiv a_1 \pmod{m_1}$$

umfassen genau die Ganzzahlen

$$x_1 = a_1 + m_1 x_2$$

für $x_2 \in \mathbb{Z}$.

Die zweite unserer Kongruenzen,

$$x \equiv a_2 \pmod{m_2},$$

kann umgeschrieben werden als

$$a_1 + m_1 x_2 \equiv a_2 \pmod{m_2},$$

was umgeschrieben werden kann als

$$m_1 x_2 \equiv a_2 - a_1 \pmod{m_2},$$

Da wir wissen, dass $\gcd(m_1, m_2) = 1$, wissen wir, dass diese Kongruenz Lösungen hat

$$x_2 \equiv (a_2 - a_1) \cdot m_1^{-1} \pmod{m_2}$$

welche sind

$$x_2 = A_1 + m_2 x_3$$

für $x_3 \in \mathbb{Z}$, und wo wir A_1 für den kleinsten positiven reduzierten Rest $(a_2 - a_1) \cdot m_1^{-1}$ modulo m_2 schreiben.

Wir wiederholen nun den Prozess:

$$x \equiv a_3 \pmod{m_3}$$

wird zu

$$a_1 + m_1(A_1 + m_2 x_3) \equiv a_3 \pmod{m_3}$$

welches wird zu

$$m_1 m_2 x_3 \equiv a_3 - a_1 - m_1 A_1 \pmod{m_3}$$

Bemerkung 3.10 Wir bemerken, dass der erste Beweis ein schönes Beispiel für einen existenziellen mathematischen Beweis ist und dass es sehr falsch ist, wenn man tatsächlich das CRT verwenden möchte, um einige gleichzeitige Gleichungen zu lösen. Das Problem mit dem ersten Beweis ist, dass das Modul M exponentiell mit der Anzahl der einzelnen Moduli m_i wächst, und der erste Beweis erfordert die Verwendung von vollständiger Arithmetik während der gesamten Berechnung.

Andererseits ist der zweite Beweis, der relativ einfach zu machen, aber ungeschickt zu formulieren ist, konstruktiv und algorithmisch und erfordert nie die Durchführung von Arithmetik größer als das Quadrat der größten der Moduli.

Beispiel 3.2 Wir verwenden das CRT, um das folgende System von Kongruenzen zu lösen.

$$x \equiv 3 \pmod{5}$$

$$x \equiv 2 \pmod{7}$$

$$x \equiv 4 \pmod{11}$$

Und wir werden dies zunächst auf die rechnerisch effiziente Weise tun.

Wir haben $x \equiv 3 \pmod{5}$, was bedeutet, dass $x = 3 + 5a$ für eine bestimmte ganze Zahl a gilt. Wir haben also

$$3 + 5a \equiv 2 \pmod{7}$$

$$5a \equiv 6 \pmod{7}$$

$$a \equiv 4 \pmod{7}$$

und somit $x = 3 + 5a = 3 + 5(4 + 7b) = 23 + 35b$ für eine bestimmte ganze Zahl b .

Wir fahren fort

$$23 + 35b \equiv 4 \pmod{11}$$

$$2b \equiv 3 \pmod{11}$$

$$b \equiv 7 \pmod{11}$$

und somit $x = 23 + 35b = 23 + 35(7 + 11c) = 268 + 385c$ für eine bestimmte ganze Zahl c .

Wir überprüfen, dass $268 \equiv 3 \pmod{5}$, dann dass $268 \equiv 2 \pmod{7}$, und schließlich dass $268 \equiv 4 \pmod{11}$; unsere Lösung

$$x \equiv 268 \pmod{385}$$

ist eindeutig modulo $385 = 5 \times 7 \times 11$.

Beispiel 3.3 Lassen Sie uns dieselbe Berechnung mit der Methode durchführen, die der Lagrange-Interpolation analog ist. Wir wollen

$$x \equiv 3 \pmod{5}$$

$$x \equiv 2 \pmod{7}$$

$$x \equiv 4 \pmod{11}$$

Wir haben $M_1 = 77$, $M_2 = 55$, $M_3 = 35$, sagen wir, und wir stellen fest, dass $b_1 \equiv 3 \pmod{5}$, $b_2 \equiv 6 \pmod{7}$, $b_3 \equiv 6 \pmod{11}$. Wir können schreiben

$$\begin{aligned} X &= 3 \times (3 \times 77) + 2 \times (6 \times 55) + 4 \times (6 \times 35) \\ &= 693 + 660 + 840 \\ &= 2193 \end{aligned}$$

und wir bemerken, dass $2193 = 268 + 5 \times 385$.

Bemerkung 3.11 Der CRT ist ein äußerst leistungsfähiges Berechnungswerkzeug. Nehmen wir an, wir haben eine Polynomgleichung $f(x) = 0$, die wir in Ganzzahlen lösen möchten. Eine Lösung in Ganzzahlen zu finden, könnte eine rechnerisch schmerzhaft Sache sein, teilweise weil man sehr schnell multipräzise Arithmetik verwenden müsste. Das Finden einer Lösung für die Kongruenz $f(x) \equiv 0 \pmod{p}$ für eine Primzahl p ist viel einfacher und würde keine Arithmetik von Werten größer als p^2 erfordern. Die Macht liegt in der Tatsache, dass der Modulus als das *Produkt* der einzelnen Primmoduli wächst, so dass, wenn der Bruchteil der möglichen Lösungen ein relativ konstanter Bruchteil der Primzahlen ist, die mögliche Lösungsmenge sehr schnell sehr dünn wird.

Wir veranschaulichen dies mit einem Beispiel. Nehmen wir an, wir wollen lösen

$$x^2 + x - 10100 = 0$$

Modulo 8, wir haben

$$0 : 0 + 0 - 4 \equiv -4 \equiv 4 \neq 0$$

$$1 : 1 + 1 - 4 \equiv -2 \equiv 6 \neq 0$$

$$2 : 4 + 2 - 4 \equiv 2 \neq 0$$

$$3 : 1 + 3 - 4 \equiv 0$$

$$4 : 0 + 4 - 4 \equiv 0$$

$$5 : 1 + 5 - 4 \equiv 2 \neq 0$$

$$6 : 4 + 6 - 4 \equiv 6 \neq 0$$

$$7 : 1 + 7 - 4 \equiv 4 \neq 0$$

und somit ist $x \equiv 3, 4 \pmod{8}$ notwendig.

Modulo 3 haben wir

$$0 : 0 + 0 - 2 \equiv -2 \equiv 1 \neq 0$$

$$1 : 1 + 1 - 2 \equiv 0$$

$$2 : 1 + 2 - 2 \equiv 1 \neq 0$$

und somit ist $x \equiv 1 \pmod{3}$ notwendig.

Modulo 5 haben wir

$$0 : 0 + 0 - 0 \equiv 0$$

$$1 : 1 + 1 - 0 \equiv 2 \neq 0$$

$$2 : 4 + 2 - 0 \equiv 1 \neq 0$$

$$3 : 4 + 3 - 0 \equiv 2 \neq 0$$

$$4 : 1 + 4 - 0 \equiv 0$$

und somit ist $x \equiv 0, 4 \pmod{5}$ notwendig.

Lassen Sie uns also unsere Lösung erstellen. (Und seien wir vorsichtig, wenn wir Gleichheiten und Kongruenzen haben.)

Beginnen Sie mit $X = 3a + 1$ für eine Variable a .

Wir benötigen $X \equiv 0, 4 \pmod{5}$

Die erste Option ist

$$X = 3a + 1 \equiv 0 \pmod{5}$$

$$3a \equiv -1 \equiv 4 \equiv 9 \pmod{5}$$

$$a \equiv 3 \pmod{5}$$

$$a = 5b + 3$$

$$X = 3a + 1 = 3(5b + 3) + 1 = 15b + 10$$

für jeden Wert von b .

Die zweite Option ist

$$X = 3a + 1 \equiv 4 \pmod{5}$$

$$3a \equiv 3 \pmod{5}$$

$$a \equiv 1 \pmod{5}$$

$$a = 5b + 1$$

$$X = 3a + 1 = 3(5b + 1) + 1 = 15b + 4$$

für jeden Wert von b .

Wir verwenden jetzt diese beiden und lösen modulo 8. Die erste Option ist

$$X = 15b + 10 \equiv 3 \pmod{8}$$

$$15b \equiv -b \equiv -7 \pmod{8}$$

$$b \equiv 7 \pmod{8}$$

$$b = 8c + 7$$

$$X = 15b + 10 = 15(8c + 7) + 10 = 120c + 115$$

für jeden Wert von c .

Die zweite Option ist

$$X = 15b + 10 \equiv 4 \pmod{8}$$

$$15b \equiv -b \equiv -6 \pmod{8}$$

$$b \equiv 6 \pmod{8}$$

$$b = 8c + 6$$

$$X = 15b + 10 = 15(8c + 6) + 10 = 120c + 100$$

für jeden Wert von c .

Die dritte Option ist

$$X = 15b + 4 \equiv 3 \pmod{8}$$

$$15b \equiv -b \equiv -1 \pmod{8}$$

$$b \equiv 1 \pmod{8}$$

$$b = 8c + 1$$

$$X = 15b + 4 = 15(8c + 1) + 4 = 120c + 19$$

für jeden Wert von c .

Schließlich ist die vierte Option

$$X = 15b + 4 \equiv 4 \pmod{8}$$

$$15b \equiv 0 \pmod{8}$$

$$b \equiv 0 \pmod{8}$$

$$b = 8c$$

$$X = 15b + 4 = 15(8c) + 4 = 120c + 4$$

für jeden Wert von c .

Modulo 120 haben wir mögliche Lösungen

$$4, 19, 100, 115$$

Wir stellen fest, dass

$$4 \equiv 1 \pmod{3}$$

$$\equiv 4 \pmod{5}$$

$$\equiv 4 \pmod{8}$$

$$19 \equiv 1 \pmod{3}$$

$$\equiv 4 \pmod{5}$$

$$\equiv 3 \pmod{8}$$

$$100 \equiv 1 \pmod{3}$$

$$\equiv 0 \pmod{5}$$

$$\equiv 4 \pmod{8}$$

$$115 \equiv 1 \pmod{3}$$

$$\equiv 0 \pmod{5}$$

$$\equiv 3 \pmod{8}$$

Mit ein wenig vorausgehender CRT-Berechnung haben wir die Brute-Force-Suche nach einer Lösung um den Faktor 30 reduziert, da wir nur vier mögliche Lösungen alle 120 Ganzzahlen testen müssen.

Wenn wir dies um eine Primzahl weiterführen würden, würden wir feststellen, dass wir benötigen

$$X \equiv 2, 4 \pmod{7}$$

Wenn wir die CRT um die Primzahl 7 erweitern würden, hätten wir nur 8 mögliche Lösungen alle 840 Ganzzahlen. Mit nur vier Anwendungen der CRT wird die Brute-Force-Suche um einen Faktor von mehr als 100 verbessert.

Und das alles wurde nur mit einer Arithmetik gemacht, die so schlecht ist wie das Quadrat der größten Primzahl im Modulus. Die Schlüsselbeobachtung ist, dass das Lösen von Kongruenzen modulo Primzahlen p ungefähr die Kosten $\mathcal{O}(\lg p)$ in Anzahl der arithmetischen Schritte hat, aber die Lösung ist garantiert modulo das Produkt der Primzahlen. Mit der Lösung für jede Primzahl p kostet $\mathcal{O}(\lg p)$ Operationen, wir *multiplizieren* den Modulus, durch den die Lösung eindeutig ist, mit all p .

3.5 Die Eulersche Totient

Definition 3.9 Die *Eulersche Phi-Funktion* auch bezeichnet als die *Totient*, ist definiert für jede positive ganze Zahl n als

$$\phi(n) = |\{a : 0 \leq a < n, \gcd(a, n) = 1\}|$$

Das heißt, $\phi(n)$ ist die Anzahl der Ganzzahlen in einer kleinsten positiven Restklasse modulo n , die zu n teilerfremd sind.

Wir stellen fest, dass $\phi(p) = p - 1$ für Primzahlen p gilt.

Satz 3.15 Die Phi-Funktion ist multiplikativ. Das heißt, wenn m und n teilerfremd sind, dann gilt

$$\phi(mn) = \phi(m)\phi(n).$$

3.6 Fermats kleiner Satz

Satz 3.16 (Fermats kleiner Satz (FLT)) Wenn p eine Primzahl ist, dann gilt für jede ganze Zahl a , dass $a^{p-1} \equiv 1 \pmod{p}$.

Wir erwähnen, dass FLT nur ein Spezialfall des Lagrangeschen Satzes ist, der in Kap. 4 vorgestellt wird. Der Lagrangesche Satz besagt, dass jedes Element in einer Gruppe, erhöht zur Ordnung der Gruppe, die Identität ist. Da die Ordnung der Gruppe der Reste modulo einer Primzahl p $p - 1$ ist, ist FLT offensichtlich ein Spezialfall von Lagrange.

Wir stellen fest, dass FLT nur in eine Richtung funktioniert: Wenn p eine Primzahl ist, dann gilt $a^{p-1} \equiv 1 \pmod{p}$ für alle a . Es gibt pathologische Zahlen n , sogenannte *Carmichael-Zahlen*, die nicht prim sind, aber für die $a^{n-1} \equiv 1 \pmod{n}$ für alle a gilt. Die kleinste und bekannteste dieser Zahlen ist 561. Es gibt eine umfangreiche Literatur über Carmichael-Zahlen. Wir werden später sehen, dass es einen Spezialfall gibt, in dem man FLT sowohl als „wenn“ als auch als „nur wenn“ verwenden kann, um einen leistungsstarken Primzahltest zu erhalten, der der Weg ist, um neue Beispiele für „die größten bekannten Primzahlen“ zu finden.

3.7 Exponentiation

Wir werden häufig benötigen um a^n für ein Element a in einer multiplikativen Struktur und eine ganze Zahl n zu berechnen. Der naive Weg zur Exponentiation besteht darin, a n -mal mit sich selbst zu multiplizieren.

Der richtige Weg zur Exponentiation ist, es binär zu machen.

Berechnen wir 3^{13} modulo 17.

Wir schreiben $13 = 8 + 4 + 1$ binär als 1101. Wir benötigen ein laufendes Produkt P , das wir auf 1 initialisieren, und einen laufenden Multiplikator M , den wir auf $a = 3$ initialisieren.

Wir schauen uns nun das rechteste Bit an. Es ist eine 1, also multiplizieren wir $P \leftarrow M \cdot P$ um $P = 3$ zu erhalten.

Wir verschieben um ein Bit nach links und quadrieren den Multiplikator $M \leftarrow M \cdot M$. Also jetzt $M = 9 = 3^2$.

Das Bit, auf das wir schauen, ist eine 0, also multiplizieren wir nicht in das laufende Produkt.

Wir verschieben um ein Bit nach links und quadrieren den Multiplikator $M \leftarrow M \cdot M$. Also jetzt $M = 81 = 13 = 3^4$.

Das Bit, auf das wir schauen, ist eine 1, also multiplizieren wir $P \leftarrow M \cdot P$ um $P = 3 \cdot 13 = 39$ zu erhalten, das wir durch 17 modulieren, um 5 zu erhalten.

Lassen Sie uns innehalten und nachdenken. Die drei rechten Bits von 13 sind 101, was 5 in Binär ist. Und wir haben als unser laufendes Produkt den Wert $3^5 = 243$, moduliert durch 17 auf 5. Dies ist der stabile Zustand; unser laufendes Produkt ist der korrekte Wert für die Potenzierung bis zu den Bits, die bisher verarbeitet wurden.

Wir verschieben um ein Bit nach links und quadrieren den Multiplikator $M \leftarrow M \cdot M$. Also jetzt $M = 169 = 16 = -1$. Das Bit, auf das wir schauen, ist eine 1, also multiplizieren wir $P \leftarrow M \cdot P$ um $P = 5 \cdot 16 = 80$ zu erhalten, das wir durch 17 modulieren, um 12 zu erhalten.

Eine Version des Python-Codes für diesen Algorithmus zur Potenzierung ist Abb. 3.1.

Es gibt eine Variation davon, bei der man die Bits von links nach rechts anstatt von rechts nach links verarbeitet. In dieser Version quadriert man das laufende Produkt, während man über die Bits des Exponenten geht. Diese Version hat den Nachteil, dass man eine Möglichkeit haben muss, auf die Bits in der Mitte eines Exponenten zuzugreifen (im Gegensatz zur rechts-nach-links-Version oben, bei der Mod-by-2 und Divide-by-2 genau das tun, was für den Umgang mit den Bits benötigt wird). Allerdings ändert sich in der links-nach-rechts-Version der Multiplikator nie. Diese Version wird in der elliptischen Kurvenkryptographie verwendet, weil elliptische Kurvengruppenoperationen teuer sind. Bei elliptischen Kurven ist das Quadrieren viel billiger als die Multiplikation, und wenn man clever ist, kann man den Multiplikator so wählen, dass die Gruppenoperation besonders einfach ist, daher ist die links-nach-rechts-Version häufiger.

```
#####
## MODULAR INTEGER EXPONENTIATION
## computes a^b mod n
def starstar(a_value, b_value, n_value):
    running_prod = 1
    multiplier = a_value
    local_exponent = b_value
    while local_exponent > 0:
        rightmost_bit = local_exponent % 2
        if rightmost_bit == 1:
            running_prod = (running_prod * multiplier) % n_value
            multiplier = (multiplier * multiplier) % n_value
            local_exponent = local_exponent // 2
    return running_prod
```

Abb. 3.1 Modulare Ganzzahlpotenzierung

3.8 Matrixreduktion

Schließlich bemerken wir eine Berechnung, die an mehreren Stellen in der modernen Kryptographie auftaucht. Betrachten Sie eine Matrixgleichung mit ganzzahligen Koeffizienten, wie zum Beispiel

$$\begin{pmatrix} 2 & 1 & 6 & 5 & 8 \\ 5 & 7 & 3 & 9 & 1 \\ 1 & 2 & 1 & 0 & 3 \\ 1 & 4 & 7 & 2 & 5 \\ 3 & 2 & 1 & 4 & 2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 37 \\ 53 \\ 0 \\ 22 \\ 25 \end{pmatrix}$$

Wenn wir dies über die reellen Zahlen lösen würden, würden wir (naiverweise) das Gaußsche Eliminationsverfahren verwenden. Über den Ganzzahlen können wir Zeilen tauschen und Zeilen zu anderen Zeilen hinzufügen, aber wir dürfen nicht teilen, es sei denn, die Division führt zu einer Zeile, die weiterhin ganzzahlige Koeffizienten hat. Wir könnten die Matrixreduktion beginnen, indem wir die erste und dritte Zeile tauschen und dann 5, 2, 1 und 3 mal die neue erste Zeile von den anderen Zeilen subtrahieren, wodurch wir eine erweiterte Matrix erhalten

$$\left(\begin{array}{ccccc|c} 1 & 2 & 1 & 0 & 3 & 0 \\ 0 & -3 & -2 & 9 & -14 & 53 \\ 0 & -3 & 4 & 5 & 2 & 37 \\ 0 & 2 & 6 & 2 & 2 & 22 \\ 0 & -4 & -2 & 4 & -7 & 25 \end{array} \right)$$

Wir können fortfahren und machen im Grunde genommen einen größten-gemeinsamen-Teiler-Prozess auf Spalte 2 und Zeilen 2 bis 5, dann Spalte 3 und Zeilen 3 bis 5, und erhalten schließlich

$$\left(\begin{array}{ccccc|c} 1 & 2 & 1 & 0 & 3 & 108 \\ 0 & 1 & 10 & 13 & -10 & -479 \\ 0 & 0 & 2 & 24 & -54 & -1032 \\ 0 & 0 & 0 & 4 & 89 & -2341 \\ 0 & 0 & 0 & 0 & 89 & -1951 \end{array} \right)$$

Die Tatsache, dass wir keine Zeilen mit Nullen bekommen, zeigt, dass die Determinante der Matrix nicht null ist und dass es eine Lösung über die reellen Zahlen zur Matrixgleichung gibt. Wir könnten jetzt versuchen, rückwärts zu lösen für ganzzahlige Werte für a, b, c, d, e , oder wir könnten weiter reduzieren, bis wir die *Hermite Normalform* erreicht haben, die eine obere Dreiecksmatrix ist, deren Koeffizienten über der Hauptdiagonalen alle kleiner (zumindest im absoluten Wert) sind als der Eintrag für diese Spalte entlang der Diagonalen.

$$\left(\begin{array}{ccccc|c} 1 & 0 & 1 & 2 & 17 & -290 \\ 0 & 1 & 0 & 1 & 82 & -1947 \\ 0 & 0 & 2 & 0 & 35 & -643 \\ 0 & 0 & 0 & 4 & 0 & -390 \\ 0 & 0 & 0 & 0 & 89 & -1951 \end{array} \right)$$

Es wird fast immer passieren, wie es hier der Fall ist, dass wir nicht rückwärts lösen und ganzzahlige Lösungen bekommen können. Alles ist jedoch nicht verloren. In den meisten Anwendungen in der Kryptographie lösen wir das System tatsächlich nicht über die Ganzzahlen, sondern modulo einer Primzahl p . In diesem Fall können wir normalerweise Lösungen für die Matrixgleichung finden; da alle Primzahlen außer 2 ungerade sind, haben wir modulo jeder ungeraden Primzahl p $d = (-195 + p)/2$ als die Lösung.

Wir werden eine Version dieser Matrixreduktion in Kap. 6 sehen, wo wir tatsächlich nur modulo 2 reduzieren werden, in Kap. 11 und 12, ebenfalls modulo 2 arbeitend, in Kap. 13, arbeitend modulo sehr großen Primzahlen, und in Kap. 15, arbeitend modulo mäßig großen Primzahlen. In einigen Anwendungen werden wir die Lösungen modulo Primzahlen verwenden, und in einigen Fällen werden wir mehr Zeilen als Spalten haben und die notwendigen Zeilen mit lauter Nullen für weitere Arbeiten verwenden.

3.9 Übungen

1. Finden Sie den größten gemeinsamen Teiler der folgenden Paare von Ganzzahlen, indem Sie eine Version des euklidischen Algorithmus verwenden.
 - a. 101 und 73
 - b. 221 und 85
 - c. 96 und 27
 - d. 152 und 86
 - e. 199 und 200
2. Finden Sie die x und y Werte des erweiterten euklidischen Algorithmus.
 - a. 101 und 73
 - b. 221 und 85
 - c. 96 und 27
 - d. 152 und 86
 - e. 199 und 200
3. Lösen Sie nach x auf:

$$23x \equiv 2 \pmod{37}$$

4. Lösen Sie nach x auf:

$$23x \equiv 2 \pmod{111}$$

5. Lösen Sie nach x auf:

$$23x \equiv 2 \pmod{343}$$

6. Finden Sie eine primitive Wurzel der Primzahlen 941, 1009 und 1013.
7. Zeigen Sie, dass wenn r eine primitive Wurzel modulo einer ungeraden Primzahl p ist, dann entweder r oder $r + p$, je nachdem, welches ungerade ist, eine primitive Wurzel von $2p$ ist.
8. Betrachten Sie die Ganzzahlen modulo $N = pq$, für verschiedene ungerade Primzahlen p und q , mit $p < q$. Zeigen Sie, dass es ein Element modulo N der Ordnung mindestens $q - 1$ geben muss. Zeigen Sie, indem Sie Beispiele für kleine Werte von p und q betrachten, dass es Elemente modulo n der Ordnung größer als $q - 1$ geben könnte.
9. Betrachten Sie die Ganzzahlen modulo $N = pq$, für verschiedene ungerade Primzahlen p und q , mit $p < q$. Zeigen Sie, dass die größtmögliche Ordnung eines beliebigen Elements modulo N das kleinste gemeinsame Vielfache von $p - 1$ und $q - 1$ ist.
10. (Programmierübung) Schreiben Sie ein Programm, das ein „endloses Sieb“ zur Erzeugung von Primzahlen ist. Bestimmen Sie eine „Blockgröße“ der Größe 10^5 , sagen wir, mit dem Index des ersten Auftretens einer Primzahl kleiner als die Blockgröße, die einen Wert im Block teilt. Dies ermöglicht es Ihrem Programm zu bestimmen, dass eine Ganzzahl im k -ten Block von Ganzzahlen k das erste Auftreten der Teilbarkeit durch eine kleine Primzahl beim Index j im Block haben wird. Dies ermöglicht es Ihnen, endlos zu sieben, wobei nur Speicher in der Größe des Blocks verwendet wird, für alle Primzahlen kleiner als das Quadrat der Blockgröße.
11. (Programmierübung) Schreiben Sie einen Code, um die kleinste primitive Wurzel modulo einer Primzahl zu berechnen, mit einer naiven Methode oder vielleicht einer naiven Methode mit einigen Verbesserungen.
12. (Programmierübung) Schreiben Sie einen Code, um den euklidischen Algorithmus und den erweiterten euklidischen Algorithmus durchzuführen.
13. (Programmierübung) Schreiben Sie einen Code, um Kongruenzen zu lösen. Stellen Sie sicher, dass Sie die Fehlerfallen einbeziehen, wenn eine Kongruenz tatsächlich nicht gelöst werden kann.

Literatur

1. D.E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, 2. Aufl. (Addison-Wesley, Boston, 1981)
2. D.H. Lehmer, Euclid's algorithm for large numbers. *Am. Math. Mon.* 227–233 (1938)
3. T. Jebelean, A generalization of the binary GCD algorithm, in *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC*, Bd. 93 (1993), S. 111–116

-
4. J. Sorenson, Two fast GCD algorithms. *J. Algorithms* **16**, 110–144 (1994)
 5. S.M. Sedjelmaci, Jebelean-Weber's algorithm without spurious factors. *Inf. Process. Lett.* **102**, 247–252 (2007)
 6. K. Weber, Parallel implementation of the accelerated GCD algorithm. *J. Symb. Comput.* **21**, 457–466 (1996)

Zusammenfassung

Die moderne Kryptographie stützt sich für ihre Fähigkeit, Klartext in Chiffretext zu verwandeln, der wie zufällige Symbolsequenzen erscheint, auf die grundlegenden Konzepte der abstrakten Algebra. In diesem Kapitel führen wir die Grundlagen von Gruppen, Ringen und Körpern ein, einschließlich Untergruppen, zyklischen Gruppen, der Ordnung von Elementen und dem Lagrangeschen Theorem. Eine *Gruppe* ist eine Menge, die unter einer Operation geschlossen ist, die normalerweise (und oft) entweder als Addition oder als Multiplikation bezeichnet wird, mit zusätzlichen Eigenschaften. Ein *Ring* hat sowohl eine Addition als auch eine Multiplikation, aber möglicherweise keine Operation, die einer Division ähnelt. Ein *Körper* hat alle Eigenschaften, die wir normalerweise mit der Durchführung von Arithmetik verbinden. Alle drei sind in gewisser Weise lediglich abstrakte Beschreibungen der gewöhnlichen Arithmetik. Beweise werden weitgehend auf später verschoben oder gar nicht durchgeführt. Und da wir mehr daran interessiert sind, Gruppen, Ringe und Körper zu verwenden, als Theoreme über sie als algebraische Objekte zu beweisen, kann dieses Kapitel weitgehend als einfache Bereitstellung von Definitionen und formalen Aussagen der Wahrheit dessen angesehen werden, was wir beobachten, wenn wir die Operationen zum Verschlüsseln und Entschlüsseln durchführen.

4.1 Gruppen

Definition 4.1 Eine *Gruppe* $G = (S, *)$ ist eine Menge S von Elementen zusammen mit einer binären Operation

$$* : S \times S \rightarrow S$$

so dass

- die Operation $*$ ist *assoziativ*, das heißt,

$$(a * b) * c = a * (b * c)$$

für alle $a, b, c \in S$;

- es gibt ein *neutrales* Element e so dass für alle $a \in S$, wir haben $a * e = e * a = a$;
- jedes Element $a \in S$ hat ein *inverses* für welches $a * a^{-1} = a^{-1} * a = e$.

Definition 4.2 Wir sagen, eine Gruppe G ist *abelsch* (oder *kommutativ*) wenn für alle Paare $a, b \in S$ wir haben $a * b = b * a$.

Wenn wir darauf bestehen, formell zu sein, schreiben wir die Gruppe als

$$G = (S, *)$$

aber oft schreiben wir einfach S und gehen davon aus, dass die Gruppenoperation bekannt ist.

Manchmal sprechen wir von der Gruppenoperation als „Multiplikation“ und manchmal sprechen wir von der Gruppenoperation als „Addition“. Wenn wir die Operation als Multiplikation betrachten, schreiben wir $a * b$ für die Operation, a^{-1} für das Inverse und 1 für die Identität. Wenn wir die Operation als Addition betrachten, schreiben wir $a + b$ für die Operation, $-a$ für das Inverse und 0 für die Identität.

Beispiel 4.1 Das klassische Beispiel für eine Gruppe wären die Ganzzahlen \mathbb{Z} unter Addition. Dies ist klarerweise eine abelsche Gruppe, mit Null als Identität und $-n$ das Inverse für jede Ganzzahl n .

Beispiel 4.2 Ein weiteres nützliches Beispiel für eine Gruppe wären die positiven Ganzzahlen modulo 16, sagen wir, unter Addition. Für den Moment weisen wir einfach darauf hin, dass für Ganzzahlen a und b , beide kleiner als n , genommen modulo n , die „Summe“ von a und b $a + b$ ist, wenn diese Summe kleiner als n ist, und $a + b - n$ ist, wenn diese Summe größer oder gleich n ist. So ist zum Beispiel 11 plus 9 genommen modulo 16

$$11 + 9 = 20 \equiv 20 - 16 = 4.$$

Wenn wir Dinge unter Verwendung von Addition modulo 16 nehmen, bilden die Ganzzahlen

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$$

klar eine abelsche Gruppe, mit Null als Identität und $16 - n$ das Inverse für jede Ganzzahl n im Bereich von 0 bis 15.

Wir bemerken, dass jede vollständige Menge von Resten funktioniert, obwohl die Angabe des Ergebnisses der Gruppenoperation etwas komplizierter wäre. Man könnte zum Beispiel die Reste mit der geringsten Größe wählen, wie

$$\{-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7\}$$

Definition 4.3 Eine *Untergruppe* einer Gruppe G ist eine Gruppe $G' = (S', *)$ so dass $S' \subseteq S$ als eine Menge von Elementen, die Identität $1_{S'}$ der Untergruppe ist die gleiche wie die Identität 1_S der Gruppe $G = (S, *)$, und das Inverse eines jeden Elements in der Untergruppe ist das gleiche wie das Inverse dieses Elements in $G = (S, *)$.

Beispiel 4.3 Ein klassisches Beispiel für eine Untergruppe wären die geraden Zahlen unter Addition, oder allgemeiner die Zahlen, die durch eine festgelegte Zahl n teilbar sind. Die Summe von zwei geraden Zahlen ist eine gerade Zahl, so dass die Abgeschlossenheit erfüllt ist, und die anderen Anforderungen werden von der gewöhnlichen Addition von Zahlen geerbt.

Beispiel 4.4 Ein weiteres Beispiel wäre die Untergruppe der Vielfachen von 4 in der Gruppe der Zahlen modulo 16. Diese Zahlen wären

$$\{0, 4, 8, 12\}$$

Definition 4.4 Eine Gruppe $G = (S, *)$ wird als *zyklisch* bezeichnet, wenn es einen *Erzeuger* g gibt, so dass für jedes Element $a \in S$ eine ganzzahlige Potenz k existiert, so dass $a = g^k$.

Dies ist die Definition in multiplikativer Notation. In additiver Notation würden wir sagen, dass es einen Erzeuger g gibt, so dass für jedes Element a ein ganzzahliges Vielfaches n existiert, so dass $a = ng$. Wenn wir mit unserem klassischen Beispiel fortfahren, stellen wir fest, dass die Zahlen unter Addition \mathbb{Z} zyklisch mit dem Erzeuger 1 sind, die geraden Zahlen ebenso zyklisch mit dem Erzeuger 2, und so weiter.

Satz 4.1 Jede Untergruppe einer zyklischen Gruppe ist zyklisch.

Beweis Lassen Sie S' die Menge der Elemente von G sein, die die Untergruppe bilden. Wir werden hier ein wenig schlampig sein und uns auf „die Gruppe S' “ nur durch ihre Elemente und nicht als Paar mit der explizit aufgeführten Operation beziehen. Wir werden dies unter Verwendung von Material beweisen, das in diesen Notizen erst in Kap. 3 über Kongruenzen erscheint. Wenn G zyklisch ist und wir G als multiplikative Gruppe schreiben, dann haben wir einen Generator g und wir können jedes Element der Gruppe mit dem Exponenten von g für dieses Element bezeichnen. Das heißt, wir können jedes Element der Gruppe als

$$\{g^0, g^1, g^2, \dots\}$$

schreiben. Lassen Sie h das Element in S' sein, das $h = g^k$ für das kleinste nichtnegative k ist. Wir behaupten, dass $S' = \{h^n | n \in \mathbb{Z}\}$. Betrachten Sie irgendein Element $t \in S'$. Wir müssen ein s haben, so dass $t = g^s$. Nach dem Divisionssatz für Ganzzahlen (den wir später erreichen werden), können wir

$$s = q \cdot k + r$$

mit r nichtnegativ und kleiner als k schreiben.

Nun, da wir $g^s \in S$ haben, und wir haben $g^k \in S$, müssen wir $g^s * g^{-qk} = g^r \in S$ haben. Daher $r = 0$; die Existenz eines nichtnegativen r würde unserer Wahl von k als dem kleinsten nichtnegativen Exponenten eines Elements in der Untergruppe widersprechen. Das bedeutet, dass $t = g^{qk} = (g^k)^q = h^q$, was wir behauptet haben; jedes Element in S kann als Potenz von h geschrieben werden, und die Untergruppe S' ist tatsächlich zyklisch. \square

Beispiel 4.5 In unserem Beispiel der Untergruppe

$$\{0, 4, 8, 12\}$$

bemerken wir, dass jedes Element in der Gruppe durch Vielfache von 4 erzeugt wird, was das 4-fache des Erzeugers der gesamten Gruppe ist.

Definition 4.5 Die *Ordnung* von eines Elements a in einer Gruppe G ist die kleinste Ganzzahl k , so dass (multiplikativ geschrieben) wir haben $a^k = 1$. Wenn eine solche Ganzzahl nicht existiert, dann wird gesagt, dass das Element eine *unendliche Ordnung* hat.

Wir stellen fest, dass, wenn die Gruppe endlich ist, die Ordnung eines Elements gut definiert ist. Da die Gruppe endlich ist, muss die Sequenz der Elemente (multiplikativ geschrieben)

$$a, a^2, a^3, \dots, a^k, \dots$$

sich wiederholen (beachten Sie, dass wir nicht behaupten, die Gruppe sei zyklisch, nur dass die Potenzen eines bestimmten Elements einen Zyklus bilden müssen). Das heißt, es gibt Exponenten i und j , so dass $a^i = a^j$. Aber dann haben wir

$$1 = a^{i-i} = a^{j-i}.$$

Wenn wir i und j als die kleinsten Ganzzahlen gewählt haben, so dass $a^i = a^j$, dann ist $j - i$ die Ordnung des Elements a .

Definition 4.6 Die *Ordnung* einer Gruppe G ist $|S|$, die Kardinalität der Menge S .

Wir stellen fest, dass es Gruppen endlicher Ordnung gibt, wie die Ganzzahlen modulo 16, von der Ordnung 16, und Gruppen unendlicher Ordnung, wie die Menge aller Ganzzahlen unter Addition.

Satz 4.2 (Lagrange) Die *Ordnung* eines beliebigen Elements einer endlichen Gruppe G teilt die Ordnung der Gruppe.

Beweis Schreiben Sie die Gruppe multiplikativ, wählen Sie ein Element a aus G und berechnen Sie seine Potenzen. Wir haben, dass

$$A = \{1, a, a^2, \dots, a^{k-1}\}$$

für eine bestimmte k , eine Untergruppe von k Elementen ist, mit $a^k = 1$. Wenn dies die Menge der Elemente in der Gruppe erschöpft, dann sind wir fertig.

Wenn nicht, wählen Sie ein beliebiges Element b in G , das nicht in A ist, und betrachten Sie

$$B = \{b, ba, ba^2, \dots, ba^{k-1}\}.$$

Keines davon sind Elemente in A , denn wenn es der Fall wäre, dass $ba^i = a^j$, dann hätten wir $b = a^{j-i} \in A$, im Widerspruch zu unserer Wahl von b .

So ist $A \cup B$ eine Menge von $2k$ Elementen in G . Wenn dies alles von G ist, sind wir fertig.

Wenn nicht, wählen Sie ein beliebiges Element c in G , das weder in A noch in B ist, und betrachten Sie

$$C = \{c, ca, ca^2, \dots, ca^{k-1}\}.$$

Keines davon sind Elemente in A oder in B . Wenn es der Fall wäre, dass $ca^i = a^j$, dann hätten wir $c = a^{j-i} \in A$. Wenn es der Fall wäre, dass $ca^i = ba^j$, dann hätten wir $c = ba^{j-i} \in B$. Beides wäre im Widerspruch zu unserer Wahl von c .

Und so weiter. Jedes Mal, wenn wir ein weiteres noch nicht gefundenes Element suchen, fügen wir tatsächlich k Elemente zu unserer Menge von Elementen in G hinzu. Da wir k Elemente auf einmal hinzufügen, muss es der Fall sein, dass wir, wenn wir alle Elemente in der Gruppe berücksichtigt haben, für ein bestimmtes *minsgesamt* km Elemente haben. Daher ist die Ordnung der Gruppe km und ist klar durch k teilbar.

Korollar 4.1 (Lagrange) Wenn G eine Gruppe mit n Elementen ist, und wir schreiben G als eine multiplikative Gruppe, dann haben wir für jedes Element $a \in G$ dass a^n gleich der Identität der Gruppe ist.

Beweis Für jedes Element $a \in G$ wissen wir, dass a^k , für ein k , das die Ordnung von a ist, die Identität ist. Wir können also schreiben

$$a^n = (a^k)^{n/k} = 1^{n/k} = 1,$$

was das Korollar beweist.

Definition 4.7 Der *Exponent* einer Gruppe G ist die kleinste ganze Zahl k , so dass $a^k = 1$ für alle Elemente $a \in G$ gilt.

4.2 Ringe

Definition 4.8 Ein *Ring* ist eine Menge S mit zwei binären Operationen, die wir Addition (+) und Multiplikation (\times) nennen, so dass

- $(S, +)$ ist eine abelsche Gruppe mit Identität 0;
- Die Multiplikation ist assoziativ, das heißt,

$$(a \times b) \times c = a \times (b \times c)$$

für alle $a, b, c \in S$;

- Die Multiplikation ist distributiv über die Addition, das heißt,

$$a \times (b + c) = (a \times b) + (a \times c)$$

und

$$(a + b) \times c = (a \times c) + (b \times c)$$

für alle $a, b, c \in S$;

Definition 4.9 Eine *multiplikative Identität* in einem Ring $R = (S, +, \times)$ ist ein von Null verschiedenes Element 1, so dass $1 \times a = a \times 1 = a$ für alle $a \in S$ gilt.

Definition 4.10 Ein Ring wird als *kommutativ* bezeichnet, wenn die Multiplikationsoperation kommutativ ist, das heißt, wenn

$$r \times s = s \times r$$

für alle $r, s \in S$ gilt.

Beispiel 4.6 Wenn wir unser klassisches Beispiel fortsetzen, stellen wir fest, dass die Ganzzahlen unter der üblichen Addition und Multiplikation einen Ring bilden. Dies ist ein kommutativer Ring mit der additiven Identität 0 und der multiplikativen Identität 1.

4.3 Felder

Wir haben nun alle Vorarbeiten erledigt und es bleibt, den algebraischen Hintergrund abzuschließen. Wir haben die gewöhnliche Arithmetik auf abstrakte Weise beschrieben, mit Addition, Multiplikation, Subtraktion (die Addition eines Inversen), Kommutativität und Distribution.

Das Einzige, was wir bei den Ganzzahlen nicht tun können und dennoch innerhalb der Menge der Ganzzahlen bleiben, ist, eine Ganzzahl durch eine andere zu teilen. Es gibt ein additives Inverses für jede Ganzzahl, aber die einzigen Ganzzahlen mit einem multiplikativen Inversen sind 1 und -1 , von denen jede ihr eigenes Inverses ist.

Um unsere Beschreibung der allgemeinen Eigenschaften der Arithmetik abzuschließen, müssen wir unsere Menge auf alle rationalen Zahlen erweitern.

Definition 4.11 Ein *Feld* ist ein Ring $R = (S, +, \times)$, für den die Multiplikation kommutativ ist und jedes von null verschiedene Element in S ein Inverses unter der Multiplikation hat. Das heißt, ein Feld ist ein Ring, der eine Gruppe unter Addition ist und für den die Elemente außer der additiven Identität eine abelsche Gruppe unter Multiplikation bilden.

Damit haben wir gerade die rationalen Zahlen \mathbb{Q} beschrieben. Unter Addition bilden die Ganzzahlen eine abelsche Gruppe. Unter Addition und Multiplikation bilden die Ganzzahlen einen Ring. Wenn wir die Menge auf alle rationalen Zahlen erweitern, können wir durch alle Ganzzahlen außer 0 teilen und innerhalb der Menge der rationalen Zahlen bleiben und erhalten so ein Feld.

Wir werden uns hauptsächlich mit endlichen Feldern beschäftigen, die aus den Ganzzahlen modulo Primzahlen p oder aus Polynomen stammen, deren Koeffizienten modulo 2 genommen werden, wobei die Polynome modulo ein bestimmtes Polynom $f(x)$ genommen werden. (Diese Konstruktionen sind die Themen der Kap. 3 und 6.) Endliche Felder werden oft als Felder $GF(p)$ im ersteren Fall oder $GF(2^n)$ im letzteren Fall bezeichnet, wobei n der Grad von $f(x)$ ist. Die Bezeichnung *GF* steht kurz für „Galois-Feld“ zu Ehren des berühmten französischen Mathematikers Évariste Galois.

4.4 Beispiele und Erweiterungen

Einer der großen Vorteile der Arbeit mit der Mathematik, die die Grundlage der Kryptographie bildet, ist, dass sie sehr konkret ist. Es gibt Beispiele.

4.4.1 Arithmetik modulo Primzahlen

Das klassische Beispiel für ein Feld ist die Menge der rationalen Zahlen. Wir werden auch großen Gebrauch von den Feldern der Ganzzahlen modulo Primzahlen machen. Wir werden diese Dinge später beweisen, aber wir werden diese Beispiele als konkrete Dinge verwenden, um die Begriffe zu verankern, die wir verwenden werden.

Satz 4.3 Sei p eine Primzahl. Die ganzen Zahlen modulo p , geschrieben als \mathbb{Z}_p , bilden ein endliches Feld unter der üblichen modularen Addition und Multiplikation. Weiterhin ist die Gruppe der ganzen Zahlen modulo p eine zyklische Gruppe unter Multiplikation.

Definition 4.12 Jeder Erzeuger der multiplikativen Gruppe der ganzen Zahlen modulo einer Primzahl p wird als *primitive Wurzel* modulo p bezeichnet.

Satz 4.4 Eine ganze Zahl n hat eine primitive Wurzel genau dann, wenn $n = 2, 4, p^k, 2p^k$, für ungerade Primzahlen p .

Eine primitive Wurzel zu finden, ist eine große Sache, und dies ist ein Fall, in dem die Theorie von der Praxis abweicht. Der derzeit beste Satz ohne Einschränkungen besagt, dass die kleinste primitive Wurzel modulo einer Primzahl p nur garantiert kleiner ist als etwa $p^{1/4}$. Mit Einschränkungen hat Victor Shoup gezeigt [1], dass unter der Annahme der verallgemeinerten Riemannschen Hypothese (an die natürlich so ziemlich jeder glaubt, dass sie wahr ist), die kleinste primitive Wurzel modulo einer Primzahl p kleiner ist als $\log^6 p$.

In der Praxis könnten wir nichts viel ausgefeilteres tun als eine naive Suche. Die ganzen Zahlen modulo p sind ein einziger Zyklus von $p - 1$ Elementen unter Multiplikation, so dass sie eine Menge bilden

$$\{1, 2 = g^{a_1}, 3 = g^{a_2}, \dots\}$$

Es stellt sich heraus, dass es eine schnelle und einfache Möglichkeit gibt zu bestimmen, ob der Exponent a_i gerade ist, und wenn der Exponent gerade ist, dann kann die ganze Zahl keine primitive Wurzel sein, weil ihre Potenzen nur die Reste von geradem Exponenten wären. Daher besteht der übliche praktische Ansatz zur Findung einer primitiven Wurzel einfach darin, mit 2, 3, 5 usw. zu beginnen, zu überprüfen, dass der Exponent nicht gerade ist, und wenn nicht, zu überprüfen, ob man durch Potenzieren früher als bei der Potenz $p - 1$ die Identität erhält.

Es ist oft der Fall in der Zahlentheorie, dass asymptotische Ergebnisse, die man beweisen kann, sehr langsam erreicht werden, so dass die Ergebnisse von Berechnungen mit relativ kleinen Zahlen irreführend sein können. Dennoch können wir bemerken, dass für die 78497 ungeraden Primzahlen kleiner als eine Million, wir 2, 3, 5, 6 und 7 als kleinste primitive Wurzeln haben, und das insgesamt 86.51% der Zeit, und nur 19 Primzahlen haben eine kleinste primitive Wurzel größer als 50. Unter den 487 Primzahlen zwischen 10^9 und $10^9 + 10000$, 85,6% haben 2, 3, 5, 6 oder 7 als kleinste primitive Wurzel.

Zum Beispiel, für $p = 11$, beobachten wir, dass 7 eine primitive Wurzel ist, mit

$$\begin{aligned} 7^1 &= 7, \\ 7^2 &\equiv 49 \equiv 5, \\ 7^3 &\equiv 35 \equiv 2, \\ 7^4 &\equiv 14 \equiv 3, \\ 7^5 &\equiv 21 \equiv 10, \\ 7^6 &\equiv 70 \equiv 4, \\ 7^7 &\equiv 28 \equiv 6, \\ 7^8 &\equiv 42 \equiv 9, \\ 7^9 &\equiv 63 \equiv 8, \\ 7^{10} &\equiv 56 \equiv 1 \end{aligned}$$

Wir beobachten, dass die Ordnung der multiplikativen Gruppe modulo einer Primzahl p ist $p - 1$ und dass, wenn wir eine primitive Wurzel haben, wir die Multiplikation durch Hinzufügen von Exponenten modulo $p - 1$ durchführen können, das heißt,

$$24 = 8 \times 3 \equiv 7^9 \times 7^4 \equiv 7^{13} \equiv 7^3 \equiv 2 \pmod{11}$$

Dies wird in der Kryptographie sehr wichtig sein. Die Verwendung der Exponenten auf diese Weise wird als *Indexkalkül* bezeichnet. Das Aufdröseln des Indexkalküls wird als *diskretes Logarithmusproblem* bezeichnet. Das heißt, wenn wir eine zyklische Gruppe G , einen Gruppengenerator g , und ein zufälliges Element $a \in G$ gegeben sind, besteht das diskrete Logarithmusproblem in G darin, den Exponenten k zu bestimmen, so dass $g^k = a$.

Wir stellen fest, dass die Sequenz der Potenzen von 7 modulo 11 ziemlich zufällig erscheint:

$$5, 2, 3, 10, 4, 6, 9, 8, 1$$

Das Problem des diskreten Logarithmus ist in der Kryptographie wichtig, weil es Gruppen gibt, wie die Ganzzahlen modulo großer Primzahlen, bei denen diese scheinbare Zufälligkeit ausgenutzt werden kann: Gegeben eine Primzahl p , einen Exponenten e und eine primitive Wurzel r , ist es rechnerisch einfach, die Potenz $s \equiv r^e \pmod{p}$ zu berechnen, aber rechnerisch schwierig, das diskrete Logarithmusproblem zu lösen, das diese Exponentiation umkehrt und e berechnet, gegeben s , r und p .

Wir stellen auch fest, dass alle Elemente $7^1, 7^3, 7^7, 7^9$, primitive Wurzeln sind. Wie man sehen kann, kann die Multiplikation von Ganzzahlen modulo p als Multiplikation von Potenzen einer primitiven Wurzel geschrieben werden. Und da

$$a^m \cdot a^n = a^{m+n}$$

nach den Regeln der Exponenten, verwandelt die Verwendung von primitiven Wurzeln ein Multiplikationsproblem modulo p in ein Problem der Addition von Exponenten modulo $p - 1$. In unserem Beispiel stellen wir fest, dass die Erzeugung eines Teilzyklus mit 7^2 bedeutet, dass wir einen Zyklus von Elementen

$$7^2, 7^4, 7^6, 7^8,$$

erhalten und dann kurzzyklisch werden, weil $7^{10} \equiv 1$. Ähnlich bedeutet die Erzeugung eines Teilzyklus mit 7^4 , dass wir einen Zyklus von Elementen

$$7^4, 7^8, 7^{12} \equiv 7^2, 7^6,$$

erhalten und dann kurzzyklisch werden, weil $7^{10} \equiv 1$. Die Tatsache, dass $7^1, 7^3, 7^7, 7^9$, primitive Wurzeln sind, liegt daran, dass ihre Exponenten relativ prim zu 10 sind.

Behalten Sie diesen Gedanken; wir werden später noch viel mehr davon machen. Tatsächlich ist die meiste Mathematik hinter der modernen Kryptographie im Grunde genommen diese Art von Multiplikationstabelle.

4.4.2 Arithmetik Modulo zusammengesetzter Zahlen

Arithmetik modulo einer Primzahl ergibt ein Feld. Arithmetik modulo einer zusammengesetzten Zahl ergibt einen Ring, kein Feld, weil nicht jedes Element ein multiplikatives Inverses hat. Elemente, die keine Inversen haben, werden *Nullteiler* genannt. Zum Beispiel, schauen wir uns Tab. 4.1 an, der Multiplikation modulo 15.

Wir beobachten, dass die Nullteiler jene Ganzzahlen sind, die Faktoren von 3 oder 5 in sich haben, und dass, wenn wir nur die acht Ganzzahlen 1, 2, 4, 7, 8, 11, 13 und 14 betrachten, die keine Faktoren von 3 oder 5 haben, wir eine Multiplikationstabelle ohne Nullteiler erhalten, wie in den Tab. 4.2 und 4.3.

Genauer gesagt, beobachten wir, dass die Multiplikation modulo $15 = 3 \times 5$ als Produkt eines 2-Zyklus und eines 4-Zyklus durchgeführt werden kann. Das heißt, es ist das Produkt eines $3 - 1 = 2$ -Zyklus und eines $5 - 1 = 4$ -Zyklus, wobei die 3 und die 5 genau die Primzahlen 3 und 5 sind, die in der Faktorisierung von 15 auftauchen. Die Arithmetik modulo 15 kann als $11^i \times 2^j$ geschrieben werden, mit $i = 0, 1$ und $j = 0, 1, 2, 3$. Im Allgemeinen, wenn $n = p \cdot q$ mit p und q Primzahlen (die in einem Moment definiert werden), dann kann die Multiplikation modulo n als Produkt eines Elements der Ordnung $p - 1$ und eines Elements der Ordnung $q - 1$ geschrieben werden. Wir werden dies später verwenden und ordnen die Multiplikationstabelle modulo 15 entsprechend als Tab. 4.3 um.

Tab. 4.1 Multiplikationstabelle modulo 15

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	2	4	6	8	10	12	14	1	3	5	7	9	11	13
3	3	6	9	12	0	3	6	9	12	0	3	6	9	12
4	4	8	12	1	5	9	13	2	6	10	14	3	7	11
5	5	10	0	5	10	0	5	10	0	5	10	0	5	10
6	6	12	3	9	0	6	12	3	9	0	6	12	3	9
7	7	14	6	13	5	12	4	11	3	10	2	9	1	8
8	8	1	9	2	10	3	11	4	12	5	13	6	14	7
9	9	3	12	6	0	9	3	12	6	0	9	3	12	6
10	10	5	0	10	5	0	10	5	0	10	5	0	10	5
11	11	7	3	14	10	6	2	13	9	5	1	12	8	4
12	12	9	6	3	0	12	9	6	3	0	12	9	6	3
13	13	11	9	7	5	3	1	14	12	10	8	6	4	2
14	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Tab. 4.2 Multiplikation mod 15, ohne Nullteiler

	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	12	6	4	2
14	14	13	11	8	7	4	2	1

Tab. 4.3 Multiplikation mod 15, umgeordnet

	1	2	4	8	11	7	14	13
1	1	2	4	8	11	7	14	13
2	2	4	8	1	7	14	13	11
4	4	8	1	2	14	13	11	7
8	8	1	2	4	13	11	7	14
11	11	7	14	13	1	2	4	8
7	7	14	13	11	2	4	8	1
14	14	13	11	7	4	8	1	2
13	13	11	7	14	8	1	2	4

Wir können diese Struktur auch auf abstrakte Weise betrachten. Betrachten Sie die Gruppe, die durch Verkettung der Symbole a und b unter der Bedingung, dass $ab = ba$, und $1a = a1 = a$, und $1b = b1 = b$, und $aa = 1$ und $bbbb = 1$ erzeugt wird. Das heißt, wir haben Elemente

- 1
- a
- b
- bb
- bbb
- ab
- abb
- $abbb$

Das heißt, wir haben das *direkte Produkt*

$$\{1, a\} \times \{1, b, bb, bbb\}$$

Wenn wir jetzt $a = 11$ (oder 7 oder 13 oder 14) und $b = 2$ (oder 8) ersetzen, ist diese abstrakte Gruppe dieselbe Gruppe wie die Gruppe der ganzen Zahlen modulo 15 unter Multiplikation.

Wir können Tab. 4.3 abstrakter als Tab. 4.4 umschreiben.

Definition 4.13 Ein *Homomorphismus* einer Gruppe (G, \circ_G) in eine Gruppe (H, \circ_H) ist eine Abbildung $f : G \rightarrow H$ so, dass wenn $g_1 \circ_G g_2 = g_3$ für zwei beliebige Elemente g_1, g_2 in der Gruppe G , dann $f(g_1) \circ_H f(g_2) = f(g_3)$ in der Gruppe H . Der Homomorphismus ist ein *Isomorphismus* wenn die Abbildung f eindeutig und auf ist.

Ein Homomorphismus von Gruppen ist also eine Abbildung, die die Gruppenoperation erhält, und die Abbildung ist ein Isomorphismus, wenn sie 1-1 und auf ist. Wir bemerken mehrere Beispiele aus dem, was wir bereits behandelt haben.

1. Lassen Sie G die Gruppe der Ganzzahlen unter Addition sein und lassen Sie $H = \{1, -1\}$ unter Multiplikation, wobei gerade Zahlen auf 1 und ungerade Zahlen auf -1 abgebildet werden. Dies ist ein Homomorphismus.
2. Lassen Sie G die Ganzzahlen unter Addition modulo einer zusammengesetzten Ganzzahl n sein, p ein Primteiler von n , und H die Ganzzahlen unter Addition modulo p . Dann ist G , das auf H durch $f(m) = m \pmod p$ abbildet, ein Homomorphismus.
3. Einer der wichtigsten Isomorphismen, die wir verwenden werden, ist der folgende. Lassen Sie G eine zyklische Gruppe von n Elementen sein, multiplikativ geschrieben, mit einem Generator g , so dass die Menge

$$\{g^0, g^1, \dots, g^{n-1}\}$$

alle Gruppenelemente auflistet. Lassen Sie H die Gruppe $\{0, 1, \dots, n-1\}$ unter Addition modulo n sein. Dann ist $f : G \rightarrow H$ definiert durch $f(g^k) = k$ ein Isomorphismus von Gruppen.

Tab. 4.4 Multiplikation mod 15, abstrahiert

	1	b	b^2	b^3	a	ab	ab^2	ab^3
1	1	b	b^2	b^3	a	ab	ab^2	ab^3
b	b	b^2	b^3	1	ab	ab^2	ab^3	a
b^2	b^2	b^3	1	b	ab^2	ab^3	a	ab
b^3	b^3	1	b	b^2	ab^3	a	ab	ab^2
a	a	ab	ab^2	ab^3	1	b	b^2	b^3
ab	ab	ab^2	ab^3	a	b	b^2	b^3	1
ab^2	ab^2	ab^3	a	ab	b^2	b^3	1	b
ab^3	ab^3	a	ab	ab^2	b^3	1	b	b^2

4. Lassen Sie $n = pq$ für Primzahlen p und q , und lassen Sie G die Ganzzahlen modulo n sein, die zu beiden p und q teilerfremd sind, unter Multiplikation. Lassen Sie

$$H = \{(r, s) : r = 1, \dots, p-1, s = 1, \dots, q-1\},$$

und definieren Sie die Gruppenoperation auf H als

$$(r, s) \circ_H (t, u) = (rt \pmod{p}, su \pmod{q})$$

Dies ist ein Isomorphismus von G und H .

4.4.3 Endliche Körper der Charakteristik 2

Ein Körper wird als *charakteristisch n* , für n eine ganze Zahl, bezeichnet, wenn es der Fall ist, dass $na = 0$ für alle a im Körper gilt. Der Körper der ganzen Zahlen modulo einer Primzahl p , beschrieben in Abschn. 4.4.1, hat genau p Elemente und ist von Charakteristik p . Wir werden später in Kap. 6 sehen, dass man für jede Primzahl p und ganze Zahl k endliche Körper, von Charakteristik p , mit p^k Elementen konstruieren kann. Wir werden dies für die Charakteristik 2 nutzen, da dies binäre Arithmetik beinhaltet, die in Computerhardware besonders effizient durchgeführt werden kann. Der neugierige Leser sollte das klassische Buch zu diesem Thema von Lidl und Niederreiter [2] konsultieren.

4.5 Übungen

1. Betrachten Sie die Gruppe der von Null verschiedenen Ganzzahlen modulo 11 unter Multiplikation. Was ist die Ordnung der Gruppe?
2. Betrachten Sie die von Null verschiedenen Ganzzahlen modulo 11 unter Multiplikation. Was ist die größte Ordnung eines Elements in der Gruppe?
3. Zeigen Sie, dass für jede Primzahl p , alle multiplikativen Gruppen G der Ordnung p zueinander isomorph sind.
4. Zeigen Sie, dass es genau zwei nicht-isomorphe Gruppen der Ordnung 4 gibt und dass beide abelsch sind. Eine davon ist die zyklische Gruppe

$$G = \{1, a, a^2, a^3\}$$

für einen Generator a , so dass a^4 die Identität ist, und die andere ist die abelsche *Klein-4-Gruppe*

$$H = \{1, a, b, ab\}$$

für die $a^2 = b^2 = (ab)^2$ die Identität ist.

5. Zeigen Sie, dass jede zyklische Gruppe abelsch ist.

6. Beweisen Sie aus der Definition und den Grundprinzipien, dass die Ganzzahlen, modulo einer Primzahl p , ein Körper unter gewöhnlicher Addition und Multiplikation bilden, wobei die additive Identität 0 und die multiplikative Identität 1 ist.
7. Beweisen Sie aus der Definition und den Grundprinzipien, dass die Ganzzahlen, modulo eines Produkts pq von Primzahlen p und q , einen Ring aber kein Körper unter gewöhnlicher Addition und Multiplikation bilden, wobei die additive Identität 0 und die multiplikative Identität 1 ist.
8. Erstellen Sie die Tabelle der kleinsten nichtnegativen Reste modulo 17 und deren Exponenten unter Verwendung von 3 als primitiver Wurzel. Erstellen Sie die Tabelle unter Verwendung von 5 als primitiver Wurzel. Finden Sie dann die Funktion, die die Exponenten auf Basis von 3 auf die Exponenten auf Basis von 5 abbildet.
9. Erweitern Sie das obige Beispiel zu einem Beweis für alle Primzahlen p : Wenn r_1 und r_2 zwei primitive Wurzeln für p sind, und wenn wir

$$r_2 \equiv r_1^i \pmod{p},$$

haben, dann haben wir

$$r_2^\ell \equiv (r_1)^{i\ell \pmod{p-1}} \pmod{p}.$$

Literatur

1. V. Shoup, Searching for primitive roots in Finite Fields. *Math. Comput.* **58**, 369–380 (1992)
2. R. Lidl, H. Niederreiter, *Introduction to Finite Fields and Their Applications*, 2. Aufl. (Cambridge University Press, Cambridge, 1997)



Quadratwurzeln und quadratische Symbole

5

Zusammenfassung

Wir werden Quadratwurzeln modulo Primzahlen mit Hilfe von primitiven Wurzeln und Exponenten berechnen. Dies unterscheidet sich etwas von der Methode, die in vielen Referenzen verwendet wird, aber wir möchten betonen, dass die Welt der additiven Exponentenarithmetik wichtig ist. Modulo einer Primzahl p arbeiten die Exponenten additiv modulo $p - 1$. Wenn wir zur RSA-Verschlüsselung kommen, bei der wir einen Modulus $N = pq$ für zwei große und unbekannte Primzahlen p und q haben, können wir nicht die gleichen Exponentenspiele wie bei Primzahlen spielen, weil $\phi(N) = (p - 1)(q - 1)$ nicht $N - 1$ ist, und es ist das $\phi(N)$, das die Arithmetik auf den Exponenten bestimmt. In den späteren Kapiteln über Faktorisierung und elliptische Kurven wird es rechnerisch vorteilhaft sein, bestimmen zu können, ob eine Zahl kongruent zu einem Quadrat modulo eines Modulus N ist oder nicht. Glücklicherweise kann festgestellt werden, ob eine Zahl ein Quadrat modulo einer Primzahl ist, oder festgestellt werden, dass eine Zahl kein Quadrat modulo einer zusammengesetzten Zahl ist, durch einen Prozess, der dem ggT ähnelt und die gleiche logarithmische Komplexität hat.

5.1 Quadratwurzeln

Satz 5.1 Sei p eine ungerade Primzahl. Sei a eine ganze Zahl, mit a nicht kongruent zu 0 modulo p . Dann hat

$$x^2 \equiv a \pmod{p}$$

entweder keine Lösungen oder zwei Lösungen modulo p .

Beweis Betrachten Sie eine primitive Wurzel g . Dann sind alle linearen Reste in der Liste

$$g, g^2, g^3, \dots, g^{p-1} \equiv 1$$

Es gibt eine ganze Zahl k , $1 \leq k \leq p-1$, so dass $a = g^k$.

Wir haben zwei Fälle.

Fall 1: Wenn k ungerade ist, dann haben wir keine Lösungen.

Es gibt keine Lösungen, weil wir für ein bestimmtes t haben müssten,

$$(g^t)^2 \equiv g^{2t} \equiv g^k$$

was implizieren würde, dass wir hätten

$$2t \equiv k \pmod{p-1}$$

Das bedeutet, dass

$$2t = k + m \cdot (p-1)$$

für eine ganze Zahl m gilt. Aber da $p-1$ gerade ist und k ungerade ist, kann dies nicht passieren.

Fall 2: Wenn $k = 2\ell$ gerade ist, dann

$$(g^\ell)^2 \equiv a$$

und

$$(-g^\ell)^2 \equiv a$$

So haben wir klarerweise mindestens zwei Lösungen.

Kann es noch andere geben?

Nun, wenn es ein m gibt, das irgendeine Lösung ist, haben wir

$$(g^m)^2 \equiv g^{2\ell} \pmod{p}$$

was impliziert, dass

$$2m \equiv 2\ell \pmod{p-1}$$

und somit, dass

$$m \equiv \ell \pmod{(p-1)/2}$$

Aber wenn wir ℓ, m so gewählt haben, dass $0 \leq \ell, m, < p-1$, dann als Ganzzahlen (und nicht nur Kongruenzen)

$$m = \ell$$

$$m = \ell + (p-1)/2$$

$$\ell = m + (p-1)/2$$

sind die einzigen Optionen, abhängig von den relativen Größen von ℓ und m . Die erste dieser und genau eine der beiden zweiten sind möglich, daher wissen wir, dass es nur die beiden Lösungen gibt. \square

Satz 5.2 Lassen Sie p eine ungerade Primzahl sein und g eine primitive Wurzel. Dann

$$g^{(p-1)/2} \equiv -1 \pmod{p}$$

Beweis Offensichtlich

$$(g^{(p-1)/2})^2 \equiv g^{p-1} \equiv 1 \pmod{p}$$

Wir wissen, dass es zwei Wurzeln von 1 gibt, nämlich ± 1 , und wir wissen aus dem vorherigen Theorem, dass es nur diese beiden Wurzeln gibt. Und wir wissen, dass

$$g^{(p-1)/2}$$

nicht $+1$ ist, weil modulo einer ungeraden Primzahl wir keine primitive Wurzel erhalten, die bis 1 hochgeht, bis wir zur $(p-1)$ -sten Potenz kommen. Daher muss es der Fall sein, dass

$$g^{(p-1)/2}$$

die andere Quadratwurzel von 1 ist und -1 ist. \square

5.1.1 Beispiele

Schauen wir uns Dinge modulo 11 und modulo 13 an.

Modulo 11, mit Exponenten in der ersten Zeile und Potenzen von 2 in der zweiten:

$$\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 4 & 8 & 5 & 10 & 9 & 7 & 3 & 6 & 1 \\ & & & & & -1 & -2 & -4 & & \end{array}$$

Modulo 13, mit Exponenten in der ersten Zeile und Potenzen von 2 in der zweiten:

$$\begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 & 7 & 1 \\ & & & & & -1 & -2 & -4 & & & & \end{array}$$

Beachten Sie die Exponenten. Sie legen das folgende Theorem nahe, dessen Beweis mit den Exponenten einfach ist.

Theorem 5.3 Sei p eine ungerade Primzahl. Dann ist -1 ein Quadrat modulo p genau dann, wenn $p \equiv 1 \pmod{4}$

Beweis Da wir wissen, dass

$$g^{(p-1)/2} \equiv -1 \pmod{p}$$

wir wissen, dass

$$x^2 \equiv -1 \pmod{p}$$

keine Lösungen hat oder Lösungen hat

$$\pm 1g^{(p-1)/4}$$

Aber jetzt

$$(p-1)/4$$

ist eine ganze Zahl, wenn und nur wenn $p \equiv 1 \pmod{4}$. □

Das ist es, was wir modulo 11 sehen, das 3 modulo 4 ist, und modulo 13, das 1 modulo 4 ist.

Offensichtlich sind die Reste, die gerade Potenzen einer primitiven Wurzel sind, Quadrate und die Reste, die ungerade Potenzen sind, sind es nicht. Da wir wissen, dass die $(p-1)/2$ -te Potenz -1 ist, wissen wir, dass -1 genau dann ein Quadrat ist, wenn $(p-1)/2$ gerade ist, was genau dann der Fall ist, wenn $p \equiv 1 \pmod{4}$ ist.

Satz 5.4 Sei p eine Primzahl, $p \equiv 3 \pmod{4}$, sei y eine ganze Zahl, und sei

$$x = y^{(p+1)/4}.$$

Dann

$$z^2 \equiv y \pmod{p}$$

hat keine Lösungen oder zwei Lösungen. Wenn es zwei Lösungen hat, sind sie $\pm x$. Wenn es keine Lösungen hat, dann

$$z^2 \equiv -y \pmod{p}$$

hat die beiden Lösungen $\pm x$.

Beweis Wir wissen, dass

$$(\pm x)^2 = y^{(p+1)/2} = y \cdot y^{(p-1)/2}$$

Wenn y eine Quadratzahl ist, dann gilt $y = g^{2k}$ für ein bestimmtes k . Daher

$$(\pm x)^2 = y^{(p+1)/2} = y \cdot (g^{2k})^{(p-1)/2} = y \cdot g^{k(p-1)} = y \cdot (g^{p-1})^k = y \cdot (1)^k = y$$

Wenn y keine Quadratzahl ist, dann gilt $y = g^{2k+1}$.

Dann

$$-y = (g^{2k+1})(g^{(p-1)/2}) = g^{2k+1+(p-1)/2}$$

Nun, da $p \equiv 3 \pmod{4}$ ist, ist $(p-1)/2$ ungerade, so ist $1 + (p-1)/2$ gerade, so ist der gesamte Exponent oben gerade. Das bedeutet, dass $-y$ das behauptete Quadrat ist. \square

Satz 5.5 Sei p eine ungerade Primzahl und a eine beliebige ganze Zahl, die nicht 0 modulo p ist. Dann gilt

$$a^{(p-1)/2} \equiv \pm 1 \pmod{p}$$

Beweis Wir wissen, dass

$$x^2 \equiv a \pmod{p}$$

Lösungen hat, wenn und nur wenn

$$a^{(p-1)/2} \equiv +1 \pmod{p}$$

Sei $a = g^k$ für ein beliebiges k , das funktioniert. Wenn also a ein Quadrat ist, dann gilt $a = g^{2m}$, und dann

$$a^{(p-1)/2} \equiv g^{m(p-1)} \equiv 1 \pmod{p}$$

Andererseits, wenn a kein Quadrat ist, dann ist es $a = g^{2m+1}$, und dann

$$a^{(p-1)/2} \equiv g^{(2m+1)(p-1)/2} \equiv g^{m(p-1)+(p-1)/2} \equiv g^{(p-1)/2} \equiv -1 \pmod{p}$$

\square

5.2 Charaktere auf Gruppen

Gegeben sei eine Gruppe $G = (S, *)$, die wir multiplikativ schreiben, ein *Charakter* auf der Gruppe ist eine Abbildung

$$\chi : S \rightarrow \mathbb{C}$$

die die Gruppenoperation erhält, das heißt, für die gilt

$$\chi(ab) = \chi(a)\chi(b)$$

wo die „Multiplikation“ innerhalb der Klammern auf der linken Seite in der Gruppe und die Multiplikation außerhalb der Klammern auf der rechten Seite in den komplexen Zahlen ist.

Wir werden die Legendre-Symbole und die Jacobi-Symbole als Charaktere auf der Gruppe der ganzen Zahlen modulo einer ganzen Zahl verwenden.

5.3 Legendre-Symbole

Sei p eine ungerade Primzahl. Das *Legendre-Symbol*

$$\left(\frac{a}{p}\right)$$

wird definiert als

$$\begin{aligned}\left(\frac{a}{p}\right) &= +1 \quad \text{if } x^2 \equiv a \pmod{p} \text{ is solvable} \\ &= -1 \quad \text{otherwise}\end{aligned}$$

Der folgende Satz zeigt unter anderem, dass das Legendre-Symbol ein Charakter auf der Gruppe der ganzen Zahlen modulo p ist.

Satz 5.6 Sei p eine ungerade Primzahl und a und b ganze Zahlen, die zu p prim sind.

1. Wenn $a \equiv b \pmod{p}$, dann

$$\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$$

2. Wenn a nicht null modulo p ist, dann

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod{p}$$

3. Wenn $ab \not\equiv 0 \pmod{p}$, dann

$$\left(\frac{a}{p}\right) \left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right)$$

4.

$$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$$

das 1 ist für Primzahlen p , die 1 modulo 4 sind, und -1 für Primzahlen, die 3 modulo 4 sind.

5.

$$\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$$

das 1 ist für Primzahlen p , die 1 oder 7 modulo 8 sind, und -1 sonst.

Beweis Wir haben im Grunde genommen alle diese außer dem letzten gemacht, und das überlassen wir dem Leser. \square

5.4 Quadratische Reziprozität

Gauss's *Gesetz der quadratischen Reziprozität* [1] macht all dies einfach und rechnerisch effizient.

Satz 5.7 Seien p und q verschiedene ungerade Primzahlen. Dann

$$\begin{aligned} \left(\frac{p}{q}\right) &= -\left(\frac{q}{p}\right) \quad \text{if } p \equiv q \equiv 3 \pmod{4} \\ &= \left(\frac{q}{p}\right) \quad \text{otherwise} \end{aligned}$$

5.5 Jacobi-Symbole

Das Legendre-Symbol ist für Primzahlen im „Nenner“ definiert. Das *Jacobi-Symbol* ist die Erweiterung durch Multiplikativität zu zusammengesetzten Zahlen im „Nenner“. Wenn wir also n haben, das sich als $n = rs$ faktorisiert, dann

$$\left(\frac{a}{n}\right) = \left(\frac{a}{rs}\right) = \left(\frac{a}{r}\right) \cdot \left(\frac{a}{s}\right)$$

Achtung Beachten Sie, dass das Auftreten des Jacobi-Symbols $+1$ *nicht* bedeutet, dass die quadratische Kongruenz lösbar ist, da es sein könnte, dass es eine gerade Anzahl von Nichtlösungen gibt, die das Produkt bilden. Ein solches Beispiel ist dieses.

$$\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right) \cdot \left(\frac{2}{5}\right) = (-1)(-1) = 1$$

Das Symbol ist $+1$, aber die quadratische Kongruenz

$$x^2 \equiv 2 \pmod{15}$$

hat keine Lösungen.

5.6 Erweitertes Gesetz der quadratischen Reziprozität

Gauss' *Gesetz der quadratischen Reziprozität* [1] erstreckt sich auf zusammengesetzte Ganzzahlen.

Satz 5.8 Seien M und N ungerade, positiv und teilerfremd. Dann

$$\begin{aligned} \left(\frac{M}{N}\right) &= -\left(\frac{N}{M}\right) \quad \text{if } M \equiv N \equiv 3 \pmod{4} \\ &= \left(\frac{N}{M}\right) \quad \text{otherwise} \end{aligned}$$

Wir bemerken auch, ohne Beweis, dass der letzte der Punkte in Theorem 5.6 auf zusammengesetzte Zahlen übertragen wird. Wenn Q ungerade ist, dann

$$\left(\frac{2}{Q}\right) = (-1)^{(Q^2-1)/8}$$

was 1 für Q ist, die 1 oder 7 modulo 8 sind, und -1 sonst.

Wir stellen fest, dass das Reziprozitätsgesetz es erlaubt, quadratische Symbole im Wesentlichen genauso schnell zu berechnen wie den größten gemeinsamen Teiler, da das Divisionsschema angewendet werden kann, um die Größe der beteiligten Ganzzahlen zu reduzieren. Zum Beispiel,

$$\begin{aligned} \left(\frac{10102}{7815}\right) &= \left(\frac{2287}{7815}\right) &= (-1)\left(\frac{7815}{2287}\right) &= (-1)\left(\frac{954}{2287}\right) \\ &= (-1)\left(\frac{2}{2287}\right)\left(\frac{477}{7815}\right) &= (-1)(+1)\left(\frac{185}{477}\right) &= (-1)\left(\frac{477}{185}\right) \\ &= (-1)\left(\frac{107}{185}\right) &= (-1)\left(\frac{185}{107}\right) &= (-1)\left(\frac{78}{107}\right) \\ &= (-1)\left(\frac{2}{107}\right)\left(\frac{39}{107}\right) &= (-1)(-1)\left(\frac{39}{107}\right) &= (-1)\left(\frac{107}{39}\right) \\ &= (-1)\left(\frac{29}{39}\right) &= (-1)\left(\frac{39}{29}\right) &= (-1)\left(\frac{10}{29}\right) \\ &= (-1)\left(\frac{2}{29}\right)\left(\frac{5}{29}\right) &= (-1)(-1)\left(\frac{5}{29}\right) &= \left(\frac{29}{5}\right) \\ &= \left(\frac{4}{5}\right) &= +1 \end{aligned}$$

Wir bemerken jedoch erneut, dass das Jacobi-Symbol $+1$ ist, aber es ist nicht der Fall, dass wir eine Lösung für

$$x^2 \equiv 10102 \pmod{7815}.$$

haben. Dies wird am einfachsten durch Faktorisierung des „Nenners“ gesehen:

$$\left(\frac{10102}{7815}\right) = \left(\frac{10102}{3}\right)\left(\frac{10102}{5}\right)\left(\frac{10102}{521}\right)$$

und wir können sehen, dass

$$\left(\frac{10102}{5}\right) = -1$$

so kann die Kongruenz nicht gelöst werden.

Wir können das oben Gesagte mit

$$\begin{aligned} \left(\frac{17194}{7815}\right) &= \left(\frac{1564}{7815}\right) = \left(\frac{4}{7815}\right) \left(\frac{391}{7815}\right) \\ &= (+1)(-1) \left(\frac{386}{391}\right) = (-1) \left(\frac{2}{391}\right) \left(\frac{193}{391}\right) \\ &= (-1) \left(\frac{5}{193}\right) = (-1) \left(\frac{193}{5}\right) \\ &= (-1) \left(\frac{3}{5}\right) = (-1)(-1) = +1 \end{aligned}$$

vergleichen. In diesem Fall, wenn wir $7815 = 3 \cdot 5 \cdot 521$ faktorisieren, sehen wir, dass

$$\begin{aligned} 17194 &\equiv 1 \pmod{3} \\ 17194 &\equiv 1 \pmod{5} \\ 17194 &\equiv 4 \pmod{521}. \end{aligned}$$

Diese haben einfache Lösungen, so können wir die Lösungen mit dem Chinesischen Restsatz

$$\begin{aligned} x &\equiv \pm 1 \pmod{3} \\ x &\equiv \pm 1 \pmod{5} \\ x &\equiv \pm 2 \pmod{521}. \end{aligned}$$

erstellen, um acht Lösungen modulo 7815 zu erhalten:

$$x \equiv 1043, 1562, 2083, 3127, 4688, 5732, 6253, 6772 \pmod{7815}$$

Wir erhalten acht Lösungen, weil die beiden Lösungen für jeden der Faktoren 3, 5, 521 unabhängig sind.

5.7 Übungen

1. Zeigen Sie, dass jedes quadratische Polynom $x^2 + ax + b$ genau zwei Wurzeln oder keine Wurzeln modulo einer ungeraden Primzahl p hat.
2. Modulo einer Primzahl p und gegeben eine primitive Wurzel r modulo p , wissen wir, dass $r^{(p-1)/2}$ ist -1 und somit ist es „die andere“ Quadratwurzel von 1 neben 1 selbst. Zeigen Sie, dass -1 modulo p selbst ein Quadrat ist, wenn und nur wenn $p \equiv 1 \pmod{4}$.

3. Indem Sie sich die Faktorisierung von $(p-1)/2$ ansehen, erklären Sie genau, wann -1 ein Würfel, eine vierte Potenz, eine fünfte Potenz usw. modulo p ist.
4. Berechnen Sie die Jacobi- und/oder Legendre-Symbole $\left(\frac{a}{b}\right)$ für die folgenden und geben Sie dann an, ob die Kongruenzen $x^2 \equiv a \pmod{b}$ lösbar sind oder nicht.
 - a. $\left(\frac{23}{59}\right)$
 - b. $\left(\frac{59}{23}\right)$
 - c. $\left(\frac{19}{39}\right)$
 - d. $\left(\frac{141}{221}\right)$
 - e. $\left(\frac{31}{55}\right)$
 - f. $\left(\frac{79}{97}\right)$

Antwort:

5. (Programmieraufgabe) Schreiben Sie Code, um quadratische Symbole zu berechnen. Beginnen Sie mit dem Legendre-Symbol und erweitern Sie es dann zum Jacobi-Symbol.

Literatur

1. I. Niven, H.S. Zuckerman, H.L. Montgomery, *An Introduction to the Theory of Numbers*, 5. Aufl. (Wiley, 1991)

Zusammenfassung

In diesem Kapitel erweitern wir uns über Ganzzahlen modulo Primzahlen hinaus, um endliche Körper der Charakteristik 2 zu betrachten. Für eine umfangreichere Darstellung endlicher Körper sollte der Leser Lidl und Niederreiter [1] konsultieren. Für eine andere Darstellung endlicher Körper der Charakteristik 2 könnte der Leser Golomb [2] konsultieren. Die Arithmetik endlicher Körper in Charakteristik 2 wird im Advanced Encryption Standard (AES) verwendet. Sie kann in anderen Kryptosystemen bevorzugt sein, da Computerhardware in Binär arbeitet und somit die zugrunde liegenden arithmetischen Operationen, die zum Verschlüsseln und Entschlüsseln benötigt werden, sehr schnell sein können.

6.1 Polynome mit Koeffizienten mod 2

6.1.1 Ein Beispiel

Wir betrachten Polynome mit Koeffizienten, die modulo einer Primzahl p genommen werden, modulo „der richtigen Art“ von Polynom des Grades n . Dies erzeugt einen endlichen Körper von p^n Elementen, deren multiplikative Gruppe von $p^n - 1$ Elementen zyklisch ist und von einem primitiven Element erzeugt wird, das ganz analog zu einer primitiven Wurzel modulo einer Primzahl ist.

Wir werden eine Kongruenzberechnung mit Polynomen durchführen, deren Koeffizienten modulo 2 genommen werden.

Wir beginnen mit

$$n_{00} = 1$$

Wir multiplizieren mit x :

$$n_{01} = x$$

und dann wieder

$$n_{02} = x^2$$

und wieder

$$n_{03} = x^3$$

Aber wir werden die Polynome modulo nehmen

$$m = 1 + x + x^3$$

was dasselbe ist (mit Koeffizienten mod 2) wie zu sagen, dass

$$1 + x \equiv x^3$$

also haben wir

$$n_{03} = x^3 \equiv 1 + x \pmod{m}$$

Wir setzen die Multiplikation mit x modulo m fort.

$$n_{04} = x + x^2$$

$$n_{05} = x^2 + x^3 \equiv 1 + x + x^2 \pmod{m}$$

Wenn wir noch einmal mit x modulo m multiplizieren, erhalten wir

$$n_{06} = x + x^2 + x^3 \equiv x + x^2 + 1 + x \equiv 1 + x^2 \pmod{m}$$

und dann multiplizieren wir noch einmal mit x und erhalten

$$n_{07} = x + x^3 \equiv x + 1 + x \equiv 1 \pmod{m}$$

und wir sind wieder am Anfang.

Die spezielle Theorie: Wir nehmen $p = 2$ weil die binäre Arithmetik besonders einfach in Computerhardware durchzuführen ist, und wir hoffen, für jeden Grad n ein Beispiel für ein „richtiges“ Polynom zu finden, so dass die zu implementierende Hardware noch besonders einfach ist.

In diesem Beispiel erzeugen die Potenzen von x modulo dem *primitiven Trinom*

$$m = 1 + x + x^3$$

des Grades 3 alle sieben nicht-all-null Bitmuster von 3 Bits Länge (was dasselbe ist wie alle sieben Nichtnull-Ganzzahlen modulo 8). Diese Bitmuster, gelesen vom niedrigsten zum höchsten Grad, sind

$$n_0 = 100$$

$$n_1 = 010$$

$$n_2 = 001$$

$$n_3 = 110$$

$$n_4 = 011$$

$$n_5 = 111$$

$$n_6 = 101$$

$$n_7 = 100$$

Das Polynom ist ein Trinom, weil es nur drei Nichtnull-Koeffizienten hat. Da der Grad 0 Term (die 1) und der Grad n Term nicht null sind, ist ein Polynom ein Trinom, wenn es nur einen Nichtnull-Koeffizienten zwischen den Termen des niedrigsten und des höchsten Grades gibt. Wir sagen, dass ein solches Polynom $f(x)$ des Grades n *primitiv* ist, wenn die Potenzen von x modulo $f(x)$, alle $2^n - 1$ unterschiedlichen Polynome des Grades kleiner oder gleich n erzeugen.

Was uns das gibt, ist eine Feldstruktur, die völlig analog zur Struktur der Ganzzahlen modulo einer Primzahl p ist, komplett mit Generatoren des multiplikativen Zyklus und der gleichen „vernünftig zufälligen“ Sequenz von Bitmustern, die das diskrete Logarithmusproblem schwierig macht.

Wir stellen weiter fest, dass wir die Bits in beide Richtungen laufen lassen können. Das heißt, wir könnten $1 + x + x^3$ oder $x^3 + x + 1$ schreiben, je nach persönlichem Geschmack in solchen Dingen, und solange wir die Dinge „auf die gleiche Weise“ in beiden Darstellungen machen, erhalten wir die gleichen Bitmuster.

6.2 Lineare Rückkopplungsschieberegister

In diesem Abschnitt werden wir beschreiben, was wahr ist, aber nicht unbedingt die Theoreme beweisen. Die Beweise und eine viel ausführlichere Diskussion dieses Materials finden Sie in Golomb [2] und in Lidl und Niederreiter [1].

Die *lineare Rückkopplungsschieberegister (LFSR)* Darstellung des ersten Beispiels erzeugt die gleiche Menge von Bitmustern, aber in einer anderen Reihenfolge.

Stellen wir uns eine anfängliche Sequenz von Bits der Länge 3 vor, betrachtet in einem Fenster der Breite 3:

$$n_0 = |100|$$

Wir fügen Bits auf der rechten Seite hinzu, indem wir die Summe mod 2 der beiden linken Bits im Fenster hinzufügen.

$$\begin{aligned}
n_0 &= |100| \\
n_1 &= 1|001| \\
n_2 &= 10|010| \\
n_3 &= 100|101| \\
n_4 &= 1001|011| \\
n_5 &= 10010|111| \\
n_6 &= 100101|110| \\
n_7 &= 1001011|100|
\end{aligned}$$

An diesem Punkt beginnt die Sequenz wieder; wir haben ein LFSR mit einer *Periode* 7.

Dies ist ein Beispiel für einen LFSR mit zwei *Taps* bei x^0 und x^1 . Das Polynom, das wir verwenden, ist immer noch $m = 1 + x + x^3$ und wir wenden dieses Polynom auf das Fenster für n_i an, um als nächstes Bit den Koeffizienten von x^3 zu erzeugen, der notwendig ist, um $m \equiv 0$ zu erzeugen.

Das heißt, der LFSR ist eine Rekurrenzrelation dritten Grades

$$x_{n+3} = x_{n+1} + x_n$$

Ein LFSR kann, nicht überraschend angesichts seines Namens, als Schieberegister mit einem XOR betrachtet werden, wie in Abb. 6.1. Wir haben ein Register von Bits mit einer anfänglichen Füllung (100, sagen wir). Das Register wird nach links verschoben, wobei das linke Bit zum Ausgang wird, und das rechte Bit wird als XOR (die Summe modulo 2) der beiden linken Bits gefüllt. Die Ausgangsbits werden also schließlich die Sequenz von Bits, die dem Fenster in n_7 oben vorausgehen.

Es ist die Schieberegisterversion, die bei Hardware-Designern beliebt ist. Register sind einfach zu bauen, Verschiebungen sind einfach, und das XOR von zwei Bits, um den leeren Platz am rechten Ende zu füllen, ist einfach. Man kann zeigen, dass die gleichen Eigenschaften der Zufälligkeit, die sich zeigen, wenn man Potenzen eines Generators modulo eines irreduziblen Polynoms nimmt (wir werden „irreduzibel“ später in diesem Kapitel definieren), in der Sequenz der Ausgangsbits eines LFSR inhärent sind (obwohl die Bitsequenz natürlich deterministisch ist und sie sich schließlich wiederholt). Aus diesem Grund wurden sie als Mittel zur Erzeugung von „zufälligen“ Bits eines Chiffrierschlüssels verwendet, die mit Klartextbits XOR-verknüpft werden, um Chiffretextbits zu erzeugen.

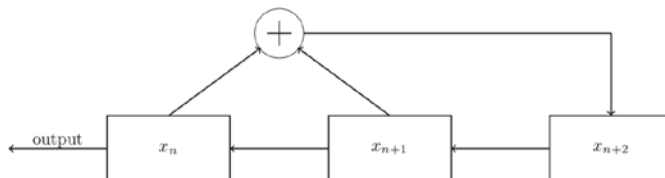


Abb. 6.1 LFSR von $x_{n+3} = x_{n+1} + x_n$

Die entscheidende Tatsache hier ist, dass wenn die Zahl $P = 2^r - 1$ eine Primzahl ist, das heißt, eine Mersenne-Primzahl ist, dann erzeugt ein LFSR vom Grad r eine Sequenz von P Bits, bevor es sich wiederholt.¹

Lassen Sie uns die Mathematik eines LFSR genauer betrachten, was mit Matrixalgebra modulo 2 gemacht werden kann.

Wenn wir eine Rekurrenzrelation dritten Grades haben, sagen wir, dann haben wir eine Gleichung

$$x_{n+3} = a_2x_{n+2} + a_1x_{n+1} + a_0x_n$$

und wir können die Rekurrenz als Matrix schreiben

$$M = \begin{pmatrix} 0 & 0 & a_0 \\ 1 & 0 & a_1 \\ 0 & 1 & a_2 \end{pmatrix}$$

mit einer 2×2 Identitätsmatrix unterhalb der Hauptdiagonalen, Nullen oberhalb der Identitätsmatrix und die Koeffizienten der Rekurrenz in der letzten Spalte. Für jedes 3-Bit-Fenster, $(x \ y \ z)$, haben wir

$$(x \ y \ z) \begin{pmatrix} 0 & 0 & a_0 \\ 1 & 0 & a_1 \\ 0 & 1 & a_2 \end{pmatrix} = (y \ z \ a_0x + a_1y + a_2z)$$

Genauer gesagt, wenn wir die Sequenz der Ausgangsbits haben, können wir diese Bits 3 auf einmal nehmen, eine Matrix daraus bilden und diese verwenden, um die Koeffizienten der Rekurrenz zu erzeugen. Wenn wir mit unserem Beispiel fortfahren und die ersten drei Fenster auswählen, haben wir

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Aus dieser Gruppe von drei Fenstern ist es trivial, die Koeffizienten zu berechnen. Aber wir könnten jede Gruppe von drei Fenstern als Matrix verwenden, wobei der Spaltenvektor auf der rechten Seite die Modulo-2-Summe ist.

So, vielleicht $x_n + x_{n+1}$ für jede Zeile für unser laufendes Beispiel.

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

¹Die ersten mehreren Mersenne-Primzahlen sind für $r = 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107$ und 127; sicherlich würden wir aus den letzten vier davon extrem lange Sequenzen von deterministisch erzeugten Bits erhalten, die zufällig die Standardtests für eine zufällige Bitsequenz erfüllen.

aus dem wir gleichzeitige Gleichungen bekommen würden

$$\begin{aligned}a_1 &= 1 \\a_1 + a_2 &= 1 \\a_0 + a_1 &= 0\end{aligned}$$

und eine eindeutige Lösung für die Koeffizienten a_i .

Dieser Ansatz funktioniert im Allgemeinen. Wenn wir eine Sequenz von Bits haben, die von einem LFSR des Grades r erzeugt wird, können wir die Matrixgleichung aufstellen

$$\begin{pmatrix} x_0 & \dots & x_{r-1} \\ \dots & & \\ x_{r-1} & \dots & x_{2r-1} \end{pmatrix} \begin{pmatrix} a_0 \\ \dots \\ a_{r-1} \end{pmatrix} = \begin{pmatrix} x_r \\ \dots \\ x_{2r} \end{pmatrix}$$

und eindeutig für die Koeffizienten lösen.

Die Frage stellt sich dann, wie wir den geeigneten Grad r der Rekurrenz bestimmen können. Dies ist auch ein Problem in der Matrixalgebra. Wenn wir die Matrix überbestimmen, indem wir vielleicht annehmen, dass wir haben sollten

$$\begin{pmatrix} x_0 & \dots & x_r \\ \dots & & \\ x_r & \dots & x_{2r} \end{pmatrix} \begin{pmatrix} a_0 \\ \dots \\ a_r \end{pmatrix} = \begin{pmatrix} x_r \\ \dots \\ x_{2r} \\ x_{2r+1} \end{pmatrix} \quad (6.1)$$

Wir werden feststellen, dass die Matrix Determinante 0 hat und das System nicht gelöst werden kann. Wir können Null-Determinanten für r , die zu klein sind, einfach durch Zufall finden, aber wenn wir wirklich nur r linear unabhängige Koeffizienten haben, dann müssen Matrizen, die zu groß sind, zu Matrizen mit Nullzeilen reduziert werden. Wenn die Determinanten der Matrizen (6.1) und größer alle Null sind für eine Sequenz von größeren Matrizen, dann ist die Größe der Matrix mit der letzten Nichtnull wahrscheinlich der geeignete Grad. Wir können überprüfen, ob dies der geeignete Grad ist, indem wir die Koeffizienten lösen und die vermeintliche Rekurrenz laufen lassen, um sie mit unserer bekannten Sequenz von Ausgangsbits zu vergleichen.

Tatsächlich können wir dies präzisieren. Betrachten Sie die linke Spalte einer Matrix wie (6.1) oder einer Matrix, die nicht nur $r \times r$ sondern sogar größer ist. Nehmen wir an, die Rekurrenz ist

$$x_{n+r} = a_{r-1}x_{n+r-1} + \dots + a_0x_n.$$

Was wir in der linken Spalte haben, ist $(x_0 \dots x_{n+r})^T$, also wenn wir die Einträge summieren, für die die a_i nicht null sind, muss die Summe 0 sein. Verschieben wir eine Spalte nach rechts, haben wir $(x_1 \dots x_{n+r+1})^T$, und die gleiche Eigenschaft muss gelten. Das heißt, das Hinzufügen der Zeilen, für die die a_i nicht null sind, ergibt genau die r -te

Zeile, also das Hinzufügen der Zeilen, für die die a_i nicht null zur r -ten Zeile sind, ergibt eine Zeile mit lauter Nullen und somit eine Null-Determinante.

Wir stellen fest, werden aber nicht beweisen, dass wir eine Korrespondenz zwischen einem LFSR des Grades r und der Periode $2^r - 1$ finden können, indem wir das charakteristische Polynom berechnen

$$\det(M + xI).$$

Im Fall unseres Beispiels wäre dies die Determinante von

$$\begin{pmatrix} x & 0 & 1 \\ 1 & x & 1 \\ 0 & 1 & x \end{pmatrix}$$

welche ist, wenn wir die Koeffizienten modulo 2 nehmen, genau das, was wir erwarten:

$$x^3 + x + 1.$$

Der eigentliche Satz ist dieser [2, S. 37].

Satz 6.1 Wenn ein LFSR ein unzerlegbares charakteristisches Polynom des Grades r hat, dann ist die Periode der Sequenz ein Faktor von $2^r - 1$. Wenn $P = 2^r - 1$ prim ist, dann entspricht jedes unzerlegbare Polynom des Grades r einem LFSR der Periode P .

Der Satz 6.1 baut auf einem Satz auf, der normalerweise früher bewiesen wird, den wir jedoch auch nicht beweisen werden.

Theorem 6.2 Jedes irreduzible Polynom modulo 2 vom Grad r teilt das Polynom

$$x^{2^r-1} + 1.$$

Wir stellen fest, dass

$$x^{2^3-1} + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Zur Vollständigkeit, als Beispiel für die beiden obigen Theoreme, stellen wir fest, dass die „andere“ Rekurrenz vom Grad 3,

$$x_{n+3} = x_{n+2} + x_n,$$

eine Matrixdarstellung hat von

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

und dass das charakteristische Polynom ist

$$\det(M + xI) = \det \begin{pmatrix} x & 0 & 1 \\ 1 & x & 0 \\ 0 & 1 & 1+x \end{pmatrix} = x^3 + x^2 + 1,$$

welches das andere irreduzible vom Grad 3 in der obigen Polynomfaktorisierung ist.

6.3 Die allgemeine Theorie

Wir werden hier nicht alles beweisen, teilweise weil die Beweise recht einfach sind. Was wir tun werden, ist darauf hinzuweisen, dass die Polynomarithmetik vollständig analog zur Arithmetik modulo gewöhnlicher Zahlen und modulo Primzahlen ist. Wir werden unsere Untersuchung auf Polynome mit Koeffizienten modulo 2 beschränken, aber Versionen dieser Ergebnisse funktionieren für Koeffizienten modulo jeder Primzahl.

Wir arbeiten modulo 2, weil die Arithmetik sehr effizient in Hardware durchgeführt werden kann, und daher *wenn* wir die mathematischen Strukturen, die wir wollen, modulo 2 zum Laufen bringen können, werden wir das tun. Arbeiten modulo ungerader Primzahlen erzeugt ähnliche mathematische Strukturen, aber dies würde Addierer und Multiplikatoren für jeden der Koeffizienten erfordern, während Arbeiten modulo 2 nur OR- und AND-Fähigkeiten erfordert. Wenn das Arbeiten in einem endlichen Feld auf der Basis von Polynomarithmetik Teil des Plans für ein kryptographisches System ist, dann wird es eine starke Präferenz für das Arbeiten modulo 2 geben, weil das die Art und Weise ist, wie die Hardware funktioniert.

Wir stellen fest, dass es einige Bedenken gibt, dass es in diesen Feldern Strukturen geben könnte, die noch nicht als ausnutzbar erkannt wurden, wenn sie in einem kryptographischen Kontext verwendet werden, aber mit diesem Vorbehalt ist klar, dass ein großer rechnerischer Vorteil daraus resultiert, mit Bits auf einem Computer zu arbeiten.

Wir stellen fest, dass die Bedingung „kleiner als“ nützlich war, zum Beispiel um zu garantieren, dass der Division Algorithmus (und somit der euklidische Algorithmus) nach einer endlichen Anzahl von Schritten endet. Der analoge Zweck wird für Polynome $f(x)$ durch den Grad des Polynoms, $\deg(f)$, erfüllt.

Satz 6.3 Der Division Algorithmus funktioniert für Polynome. Das heißt, für jedes zwei Polynome $f(x)$ und $g(x)$ gibt es Polynome $q(x)$ und $r(x)$ so dass

$$f(x) = q(x) \cdot g(x) + r(x),$$

mit $\deg(r) < \deg(g)$.

Satz 6.4 Der euklidische Algorithmus funktioniert für Polynome.

Definition 6.1 Ein Polynom $p(x)$ wird *irreduzibel* oder *prim* genannt, wenn es einen positiven Grad hat und wenn $p(x) = a(x)b(x)$ impliziert, dass eines von $a(x)$ und $b(x)$

eine Konstante ist. (Und da wir mit Polynomen mit Koeffizienten mod 2 arbeiten, müsste diese Konstante 1 sein.)

Satz 6.5 Wenn wir für ein irreduzibles Polynom $p(x)$ haben, dass $p(x) \mid a(x)b(x)$, dann entweder $p(x) \mid a(x)$ oder $p(x) \mid b(x)$.

Satz 6.6 Der Ring der Polynome mit Koeffizienten mod 2, modulo einem Polynom $m(x)$, ist ein Feld, wenn und nur wenn $m(x)$ irreduzibel ist.

Zur Bequemlichkeit des Lesers kreuzen wir die Terminologie des Feldes der Reste modulo Primzahlen und der endlichen Felder der Charakteristik 2.

Für Primzahlen:

- Wir sagen, eine ganze Zahl p ist prim, wenn es keine Teiler von p gibt, außer 1 und p .
- Wenn eine ganze Zahl p prim ist, dann bilden die kleinsten positiven linearen Reste $0, 1, \dots, p-1$ ein Feld von p Elementen unter modularer Addition und Multiplikation. Die multiplikative Gruppe der nicht-null linearen Reste ist der Ordnung $p-1$ und kann als Potenzen modulo p einer primitiven Wurzel erzeugt werden, und die Multiplikation modulo p ist das gleiche wie die Addition der Exponenten einer festen primitiven Wurzel. Die primitiven Wurzeln modulo p sind die kleinsten linearen Reste, deren Exponenten, als Potenzen einer primitiven Wurzel, relativ prim zu $p-1$ sind.

Für Polynome:

- Wir sagen, dass ein Polynom $f(x)$, mit Koeffizienten 0, 1 modulo 2, primitiv ist, wenn alle Polynome modulo $f(x)$, und mit Koeffizienten modulo 2, als Potenzen von x erzeugt werden können.
- Wir sagen, dass ein Polynom $p(x)$, mit Koeffizienten modulo 2, prim ist, oder (äquivalent) irreduzibel, wenn kein Schreiben $p(x) = a(x)b(x)$ als Polynome ohne entweder $a(x) = 1$ oder $b(x) = 1$ gemacht werden kann.
- Der Ring der Polynome mit Koeffizienten modulo 2 und modulo einem Polynom $f(x)$ ist ein Feld, wenn und nur wenn $f(x)$ irreduzibel ist.

In den Tab. 6.1, 6.2, 6.3, 6.4 und 6.5 sind die irreduziblen Polynome $f(x)$ kleinen Grades zusammen mit dem kleinsten Generator dargestellt. Da die Potenzen von x die gesamte Menge der Reste von $f(x)$ erzeugen, ist ein primitives Polynom notwendigerweise irreduzibel. Wir stellen fest, dass wenn ein Polynom irreduzibel ist, dann ist auch sein Umkehrpolynom irreduzibel, so dass wir von den Paaren von Polynomen, die wir auflisten könnten, dasjenige mit den wenigsten Nichtnull-Koeffizienten präsentieren. Weitere Tabellen finden sich bei Zierler und Brillhart [3, 4].

Tab. 6.1 Grad 2, 3, 4
Irreduzibles und kleinste
Generatoren

0	1	2	3	4		0	1
1	1	1				0	1
1	1	0	1			0	1
1	0	1	1			0	1
1	1	1	1	1		1	1
1	1	0	0	1		0	1
1	0	0	1	1		0	1

Tab. 6.2 Grad 5 Irreduzibles
und kleinste Generatoren

0	1	2	3	4	5	0	1
1	1	1	1	0	1	0	1
1	1	1	0	1	1	0	1
1	1	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	0	1	0	0	1	0	1
1	0	0	1	0	1	0	1

Tab. 6.3 Grad 6 Irreduzibles
und kleinste Generatoren

0	1	2	3	4	5	6	0	1
1	1	1	0	1	0	1	1	1
1	1	1	0	0	1	1	0	1
1	1	0	1	1	0	1	0	1
1	1	0	0	1	1	1	0	1
1	1	0	0	0	0	1	0	1
1	0	1	1	0	1	1	0	1
1	0	1	0	1	1	1	1	1
1	0	0	1	0	0	1	1	1
1	0	0	0	0	1	1	0	1

6.4 Normale Basen

Es sollte niemanden überraschen, der Berechnungen durchführt, dass „gewöhnliche“ Computer nicht für den Zweck der Beschleunigung von Berechnungen in der Zahlentheorie konzipiert sind. Computer, die für ernsthafte Berechnungen vorgesehen sind, zielen unweigerlich auf Gleitkomma-Berechnungen ab, normalerweise zum Lösen

Tab. 6.4 Grad 7 Irreduzibles
und kleinste Generatoren

0	1	2	3	4	5	6	7	0	1
1	1	1	1	1	1	0	1	0	1
1	1	1	1	0	1	1	1	0	1
1	1	1	1	0	0	0	1	0	1
1	1	1	0	1	1	1	1	0	1
1	1	1	0	0	1	0	1	0	1
1	1	0	1	0	1	0	1	0	1
1	1	0	1	0	0	1	1	0	1
1	1	0	0	1	0	1	1	0	1
1	1	0	0	0	0	0	1	0	1
1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	0	0	1	0	1
1	0	1	0	1	0	1	1	0	1
1	0	1	0	0	1	1	1	0	1
1	0	0	1	1	1	0	1	0	1
1	0	0	1	0	0	0	1	0	1
1	0	0	0	1	1	1	1	0	1
1	0	0	0	1	0	0	1	0	1
1	0	0	0	0	0	1	1	0	1

von Differentialgleichungen, zum Lösen von Problemen in der linearen Algebra oder für Monte-Carlo-Simulationen, die Zufallszahlen erfordern. Ganzzahlarithmetik ist normalerweise sehr viel eine sekundäre Priorität, und Arithmetik in endlichen Feldern modulo Polynome wird kaum beachtet. Aus diesem Grund ist es in der diskreten Mathematik und in der Zahlentheorie, einschließlich der Mathematik, die die Kryptographie unterstützt, üblich, Algorithmen und Darstellungen zu entwickeln, die die Berechnung erleichtern.

In diesem Kapitel haben wir die Elemente eines endlichen Feldes $GF(2^n)$ auf die übliche Weise als Polynome dargestellt, mit Koeffizienten von 0 oder 1, die die Potenzen von x multiplizieren. Für Berechnungszwecke gibt es eine alternative Darstellung, die eine effizientere Berechnung ermöglicht. Nach Mullin et al. [5] schreiben wir das Feld in Bezug auf eine normale Basis um und dann in Bezug auf eine optimale normale Basis.

Definition 6.2 Eine *normale Basis* für ein endliches Feld $GF(2^n)$ ist eine Basis

$$N = \{\beta, \beta^2, \beta^4, \dots, \beta^{2^{n-1}}\}$$

Tab. 6.5 Grad 8 Irreduzibles und kleinste Generatoren

0	1	2	3	4	5	6	7	8	0	1	2	3
1	1	1	1	1	1	0	0	1	1	1		
1	1	1	1	1	0	1	0	1	0	1		
1	1	1	1	1	0	0	1	1	1	1		
1	1	1	1	0	0	1	1	1	0	1		
1	1	1	0	1	1	1	0	1	1	1		
1	1	1	0	1	0	1	1	1	1	1	1	
1	1	1	0	0	1	1	1	1	0	1		
1	1	1	0	0	0	0	1	1	0	1		
1	1	0	1	1	1	1	0	1	1	0	0	1
1	1	0	1	1	0	0	0	1	1	1		
1	1	0	1	0	1	0	0	1	0	1		
1	1	0	1	0	0	0	1	1	0	1	1	
1	1	0	0	1	1	1	1	1	0	1	1	
1	1	0	0	0	1	1	0	1	0	1		
1	1	0	0	0	0	1	0	1	1	1		
1	1	0	0	0	0	1	1	1	0	1		
1	0	1	1	1	1	0	1	1	1	1	1	
1	0	1	1	1	0	1	1	1	0	1	1	
1	0	1	1	1	0	0	0	1	0	1		
1	0	1	1	0	1	0	0	1	0	1		
1	0	1	1	0	0	1	0	1	0	1		
1	0	1	1	0	0	0	1	1	0	1		
1	0	1	0	1	1	1	1	1	0	1		
1	0	1	0	0	1	1	0	1	0	1		
1	0	0	1	1	1	1	1	1	1	1		
1	0	0	1	1	1	0	0	1	1	1		
1	0	0	1	0	1	1	0	1	0	1		
1	0	0	1	0	1	0	1	1	0	1		
1	0	0	0	1	1	1	0	1	0	1		
1	0	0	0	1	1	0	1	1	0	1	1	

so dass jedes Element α von $GF(2^n)$ als lineare Kombination

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}$$

mit a_i entweder 0 oder 1 für alle i geschrieben werden kann.

Jedes endliche Feld $GF(2^n)$ hat eine normale Basis [1], und die Verwendung einer normalen Basis ist eine Übung in linearer Algebra. Wir stellen fest, dass dies wirklich nur eine spezielle Variation der Wahl eines primitiven Generators für die multiplikative Gruppe eines Feldes ist. Wenn man nur an der Geschwindigkeit der Berechnung im Feld der Reste modulo einer sehr großen Primzahl interessiert wäre, von 1024 Bits zum Beispiel, dann hätte die Verwendung von 2 als primitiver Wurzel den Vorteil, dass die Multiplikation eine Bitverschiebung wäre, normalerweise gefolgt von einer Subtraktion.

Ohne auf die Details der linearen Algebra einzugehen, stellen wir fest, dass eine normale Basis es uns ermöglicht, mit den Koeffizienten zu spielen und so die Berechnung einfacher und schneller zu machen, insbesondere wenn sie, wie in diesem Fall, mit binären Koeffizienten durchgeführt wird, für die die Computerhardware natürlich geeignet ist.

Wir erinnern uns an den klassischen Algorithmus zur Exponentiation, der von Knuth [6, S. 441ff.] den Arabern zugeschrieben wird, für den wir Python-Code in Kap. 3 hatten. Um a^e zu berechnen, schreiben wir e in Binärform. Wir behalten einen laufenden Multiplikator m und ein laufendes Produkt p , mit m initialisiert zu a und p initialisiert zu 1. Beim Lesen der Bits von e von rechts nach links ersetzen wir p durch $p * a$ wenn das Bit eine 1 ist, tun nichts wenn das Bit eine 0 ist, und dann bewegen wir uns in den Bits von e nach links während wir den laufenden Multiplikator quadrieren $m = m * m$.

Exponentiation ist ein entscheidender Teil der modernen Kryptographie, und das erste, was wir über Exponentiation sagen können, ist, dass das Quadrieren in der Mitte der Iteration besonders einfach ist, wenn das endliche Feld in einer normalen Basis geschrieben ist: Wenn

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}$$

welches wir einfacher schreiben können als

$$A = (a_0, a_1, \dots, a_{n-1}),$$

dann

$$A^2 = \sum_{i=0}^{n-1} a_i^2 \beta^{2^{i+1}},$$

und, da wir modulo 2 arbeiten, ist dies

$$A^2 = \sum_{i=0}^{n-1} a_i \beta^{2^{i+1}},$$

welches wir einfacher schreiben als

$$A = (a_{n-1}, a_0, a_1, \dots, a_{n-2}).$$

Das heißt, das Quadrieren eines Elements im endlichen Feld kann mit einer zirkulären Verschiebung der Bits, die das Element in einer normalen Basis repräsentieren, durchgeführt werden. Koeffizienten für allgemeine Terme A^{2^k} sind ähnlich nur zirkuläre Bitverschiebungen, und wir stellen fest, dass die Arithmetik tatsächlich einfach ist, da modulo 2 wir haben $x = x^2$ für $x = 0, 1$, und die Kreuzprodukte verschwinden.

Man kann die Arithmetik weiter vereinfachen, um alle Produkte von Elementen im endlichen Feld durch Koeffizientenverschiebung zu ermöglichen, indem man eine *optimale normale Basis* [5] annimmt, die für alle endlichen Felder $GF(2^k)$ relevant für die Kryptographie existiert.

6.5 Übungen

In all diesen Übungen gehen wir davon aus, dass die Koeffizienten einfach 0 und 1 sind.

1. Listen Sie die irreduziblen Polynome vom Grad 4 auf.
2. Listen Sie die primitiven Polynome vom Grad 4 auf.
3. Geben Sie eine Darstellung für den endlichen Körper $GF(2^5)$ an.
4. Geben Sie eine normale Basis für den endlichen Körper $GF(2^5)$ an.
5. Es gibt drei Polynome, die zur Erzeugung des endlichen Körpers $GF(2^4)$ verwendet werden können. Zwei davon sind primitiv: $x^4 + x + 1$ und $y^4 + y^3 + 1$. Laut Theorem sind diese Körper isomorph. Geben Sie die explizite Isomorphie an, die einen Körper auf den anderen abbildet.
6. Ihnen wird die Sequenz von Bits

1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0

gegeben, von denen Sie vermuten, dass sie von einem LFSR erzeugt wurden. Finden Sie die Rekurrenz. (Die Leerzeichen sind nur zur Verbesserung der Lesbarkeit vorhanden.)

7. (Programmieraufgabe) Schreiben Sie Programme zur Unterstützung bei der Lösung von Problem 6. Diese würden hauptsächlich Matrixreduktionen über den Integers modulo 2 sein.

Literatur

1. R. Lidl, H. Niederreiter, *Introduction to Finite Fields and Their Applications*, 2. Aufl. (Cambridge University Press, Cambridge, 1997)
2. S. Golomb, *Shift Register Sequences* (Aegean Park Press, 1982)
3. N. Zierler, J. Brillhart, On primitive trinomials (mod 2). Inform. Control **13**, 541–554 (1968)

-
4. N. Zierler, J. Brillhart, On primitive trinomials (mod 2) (part 2). Inform. Control **14**, 566–569 (1969)
 5. R.C. Mullin, I.M. Onyschuk, S.A. Vanstone, R.M. Wilson, Optimal normal bases in $GF(p^n)$. Discrete Appl. Math. 22, 149–161 (1989)
 6. D.E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, 2. Aufl. (Addison-Wesley, 1981)

Zusammenfassung

Elliptische Kurven sind eines der eleganteren Objekte in der Algebra. Ein Hintergrundwissen über die zugrunde liegende Arithmetik von Kurven ist für die Kryptographie notwendig, da sie für (mindestens) drei Zwecke verwendet werden: zum Faktorisieren von Ganzzahlen, für die Kryptographie selbst und für den Schlüsselaustausch. Die Arithmetik der elliptischen Kurven ähnelt in vielerlei Hinsicht der Arithmetik der Ganzzahlen modulo Primzahlen oder zusammengesetzten Zahlen, zumindest. In diesem Kapitel präsentieren wir eine Einführung in elliptische Kurven als Hintergrund für spätere Kapitel, die sie für kryptographische Zwecke verwenden.

7.1 Grundlagen

Definition 7.1 Eine *elliptische Kurve* über einem Feld K ist die Menge der Punkte, die eine Gleichung mit ganzzahligen Koeffizienten erfüllen, die quadratisch in einer Variablen und kubisch in einer anderen ist.

$$\mathcal{E} : y^2 = x^3 + ax + b$$

Eine ausgezeichnete erweiterte Referenz für das Material in diesem Kapitel ist [1].

Wir werden sehen, dass die Punkte auf einer Kurve eine Gruppe bilden. Zu Beginn dieses Kapitels werden wir uns nur mit Punkten auf \mathcal{E} im Feld der rationalen Zahlen \mathbb{Q} beschäftigen, werden dann aber zu Punkten übergehen, die in Feldern von ganzen Zahlen modulo einer großen Primzahl P und endlichen Feldern $GF(2^k)$ der Charakteristik 2 liegen. Wir werden auch Punkte auf einer Kurve betrachten, die aus dem Ring der ganzen Zahlen modulo einer zusammengesetzten Zahl N stammen. In diesem Fall sind die Mechanismen zur Berechnung der Gruppenoperation die gleichen, aber die Punkte

modulo einer zusammengesetzten Zahl zu nehmen, hat einen ähnlichen Effekt wie das Arbeiten mit Nullteilern im Ring der ganzen Zahlen modulo N .

Bemerkung 7.1 Im Allgemeinen könnten wir denken, dass wir die allgemeinere Form

$$\mathcal{E}' : y^2 + ay = x^3 + bx^2 + cx + d$$

berücksichtigen müssten. Allerdings können wir den linearen Term in y und den quadratischen Term in x loswerden, vorausgesetzt wir können durch 2 und 3 teilen. Da wir zunächst an *rationalen* Zahlensystemen interessiert sind, mit $K = \mathbb{Q}$, können wir substituieren

$$y = y' - a/2$$

und

$$x = x' - b/3$$

und erhalten ein rational äquivalentes Polynom ohne den linearen y -Term oder den quadratischen x -Term. Wir werden gleich den Prozess definieren, durch den wir die Punkte auf einer Kurve als Gruppe behandeln, und man kann leicht zeigen, dass die Gruppe der Punkte auf \mathcal{E}' mit rationalen Koordinaten isomorph zur Gruppe der Punkte auf \mathcal{E} mit rationalen Koordinaten ist.

Später werden wir uns mit Kurven beschäftigen, deren Punkte nicht aus den rationalen Zahlen, sondern aus Elementen eines endlichen Feldes stammen, deren Koeffizienten mod 2 genommen werden. In diesen Fällen können wir nicht durch 2 teilen; daher könnten wir einen linearen Term in y haben.

Bemerkung 7.2 Dies ist nur eine von mehreren „Standard“-Darstellungen für die Kurve. Wir sehen auch manchmal die Weierstrass-Form

$$\mathcal{E}'' : y^2 = 4x^3 - g_2x - g_3$$

Die mathematischen Strukturen sind unabhängig von der Formulierung gleich. Es sind nur die algebraischen Formeln auf Highschool-Niveau, die neu gemacht werden müssen.

Eine elliptische Kurve hat oft ein Diagramm wie das in Abb. 7.1 gezeigte. Dies ist wirklich die Ebene, die durch eine dreidimensionale Oberfläche bei einem bestimmten Wert der z -Koordinate schneidet, und die allgemeine Oberfläche ähnelt der eines Sattels. Wenn man in z nach unten drückt, verlängert sich die isolierte Ovale und verbindet sich schließlich mit dem gekrümmten Abschnitt auf der rechten Seite.

7.1.1 Gerade Linien und Schnittpunkte

Da dies eine Kubik in x ist, sollte eine gerade Linie den Graphen der Kurve in drei Punkten schneiden (Vielfachheiten mitgezählt). Das heißt, wenn wir die Linie

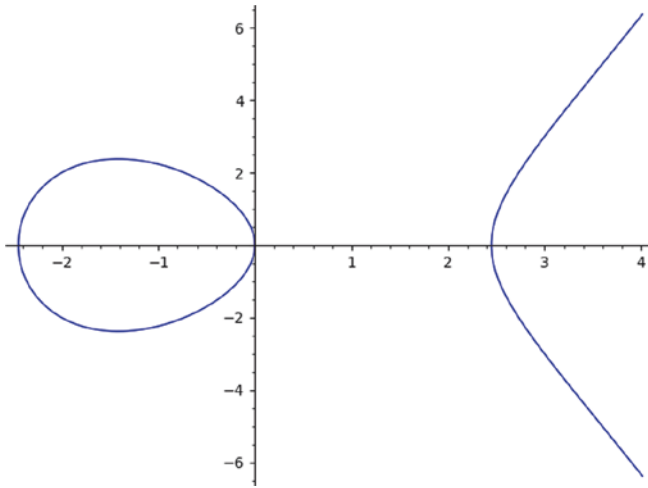


Abb. 7.1 Graph von $y^2 = x^3 - 6x + 9$

$$y = Mx + B$$

mit der Kurve schneiden, erhalten wir

$$M^2x^2 + 2MBx + B^2 = x^3 + ax + b$$

was ist

$$x^3 - M^2x^2 + (a - 2MB)x + b - B^2 = 0$$

und wenn wir die Kubik faktorisieren, erhalten wir

$$\begin{aligned} 0 &= (x - x_1)(x - x_2)(x - x_3) \\ &= x^3 - (x_1 + x_2 + x_3)x^2 + x(*) - x_1x_2x_3 \end{aligned}$$

wo wir uns wirklich nicht um die linearen Terme kümmern. Was uns wirklich interessiert, sind die Newton-Gleichungen

$$\begin{aligned} x_1 + x_2 + x_3 &= M^2 \\ x_1x_2x_3 &= B^2 - b. \end{aligned}$$

Daher, gegeben zwei Punkte (x_1, y_1) und (x_2, y_2) mit *rationalen* Koordinaten, dann

$$M = \frac{y_2 - y_1}{x_2 - x_1}$$

ist rational, und so ist

$$B = y_1 - x_1 \cdot \left(\frac{y_2 - y_1}{x_2 - x_1} \right)$$

Da x_1 rational ist, x_2 ist rational, und M ist rational, x_3 muss auch rational sein. Und da x_3 rational ist, und B rational ist, haben wir, dass y_3 rational ist.

Das heißt, zwei Punkte mit rationalen Koordinaten bestimmen eine gerade Linie, mit rationaler Steigung und Achsenabschnitt, die die Kurve in einem dritten Punkt schneidet, der ebenfalls rationale Koordinaten hat. Wir stellen fest, dass dies nicht nur eine Eigenschaft von rationalen Zahlen ist; wenn wir Lösungen betrachten, deren Koordinaten in irgendeinem Feld liegen, erhält man das gleiche Ergebnis: Da die Arithmetik in einem Feld eine wohldefinierte Division hat, wird auch der dritte Punkt, der eine gerade Linie schneidet, die zwei Punkte mit Koordinaten im Feld verbindet, ebenfalls Koordinaten im Feld haben.

7.1.2 Tangentenlinien

Die obigen Formeln funktionieren für die Betrachtung der Geraden, die durch zwei verschiedene Punkte bestimmt wird. Was ist mit zwei Punkten, die der gleiche Punkt sind?

Dafür benötigen wir die Tangente, also differenzieren wir.

$$2yy' = 3x^2 + a$$

und somit

$$y' = \frac{3x^2 + a}{2y}$$

was besagt, dass die Steigung M der Tangentenlinie an einem Punkt (x_1, y_1) ist

$$M = \frac{3x_1^2 + a}{2y_1}$$

und somit dass

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1} \right)^2 - 2x_1$$

Wiederum, wenn wir mit rationalen Koordinaten beginnen, dann hat der dritte Punkt rationale Koordinaten.

7.1.3 Formeln

Wir verwenden die Kurve

$$\mathcal{E} : y^2 = x^3 + ax + b$$

Wenn wir zwei Punkte mit rationalen Koordinaten haben

$$P_1 = (x_1, y_1)$$

$$P_2 = (x_2, y_2)$$

dann bestimmen sie eine Linie

$$y = Mx + B$$

mit

$$M = \frac{3x_1^2 + a}{2y_1}$$

oder

$$M = \frac{y_2 - y_1}{x_2 - x_1}$$

abhängig davon, ob $P_1 = P_2$ oder nicht, und

$$B = y_1 - Mx_1$$

Nach den Newtonschen Gleichungen für die Wurzeln haben wir, dass die Gerade die Kurve in einem dritten Punkt schneidet

$$P_3 = (x_3, y_3)$$

mit

$$x_3 = M^2 - x_1 - x_2.$$

Dies reicht aus, um y_3 bis auf das Vorzeichen zu bestimmen, und wir können das Vorzeichen bestimmen durch

$$M = \frac{y_3 - y_1}{x_3 - x_1}.$$

Wir bemerken, dass diese Formel einen Wert für y_3 bestimmt, der die y -Koordinate des dritten Punktes auf der Geraden ist. Im nächsten Unterabschnitt werden wir diese y -Koordinate negieren, indem wir den Punkt über die x -Achse spiegeln, um die Struktur einer additiven Gruppe auf den rationalen Punkten einer elliptischen Kurve zu legen.

7.1.4 Die Mordell-Weil Gruppe

Die oben beschriebene Methode, die als Sehnen- und Tangentenmethode bezeichnet wird, ist seit langem bekannt als eine Methode, um weitere Punkte auf einer Kurve mit rationalen Koordinaten zu finden, vorausgesetzt, es gibt mindestens einen Punkt auf der

Kurve mit rationalen Koordinaten. Es war ein bedeutendes Ergebnis von Louis Mordell, das 1922 veröffentlicht wurde, dass die Punkte auf einer elliptischen Kurve, unter dem Sehnens- und Tangentenprozess, eine abelsche Gruppe bilden, die von einer endlichen Anzahl von Basispunkten erzeugt wird.

Das Gruppengesetz, das normalerweise additiv geschrieben wird, besagt, dass

1. die Identität der Gruppe der „Punkt im Unendlichen“ ist.
2. das Negative (Gruppeninvers) eines Punktes $P = (x, y)$ ist der Punkt $-P = (x, -y)$, der die Spiegelung an der x -Achse ist.
3. die drei Punkte auf einer geraden Linie summieren sich zur Identität.

Der letzte Punkt bedeutet, dass wenn wir drei kollineare Punkte haben

$$P_1 = (x_1, y_1), \quad P_2 = (x_2, y_2), \quad P_3 = (x_3, y_3)$$

dann haben wir in der Gruppe, dass

$$P_1 + P_2 = -P_3 = (x_3, -y_3)$$

Der Leser sollte sehr genau darauf achten, ob wir y_3 oder $-y_3$ in den verschiedenen Formeln wollen. Dies ist wichtig und leicht falsch zu verstehen, wenn man nicht vorsichtig ist.

Beispiel 7.1 Lassen Sie

$$y^2 = x^3 - 36x$$

und betrachten Sie die Punkte

$$P = (-3, 9)$$

$$Q = (-2, 8)$$

Addieren $P + Q$:

Für $P + Q$ erhalten wir

$$M = \frac{9 - 8}{-3 + 2} = \frac{1}{-1} = -1$$

also

$$x_3 = 1 + 3 + 2 = 6$$

In diesem Fall müssen wir uns keine Sorgen um das Vorzeichen von y_3 machen, weil

$$y_3^2 = 216 - 216 = 0$$

also haben wir

$$y_3 = 0$$

und

$$P + Q = (6, 0)$$

Verdoppeln P :

Um $2P$ zu berechnen, benötigen wir

$$y' = \frac{3x^2 - 36}{2y}$$

und somit

$$M = \frac{27 - 36}{18} = \frac{-1}{2}$$

aus dem wir erhalten

$$x_3 = \frac{1}{4} + 3 + 3 = \frac{25}{4}$$

und dann

$$\frac{-y_3 - 9}{25/4 + 3} = \frac{-1}{2}$$

$$-y_3 = 9 + \frac{-1}{2} \cdot \frac{37}{4} = \frac{72}{8} - \frac{37}{8} = \frac{35}{8}$$

also

$$2P = (25/4, -35/8)$$

Beispiel 7.2 Nun lassen Sie

$$y^2 = x^3 + 1$$

und betrachten Sie den Punkt

$$P = (2, 3)$$

Dann für $2P$ haben wir

$$M = 3x^2/2y = 12/6 = 2$$

und somit

$$x_3 = 4 - 2 - 2 = 0$$

Dies bedeutet $y_3 = \pm 1$ und wir bestimmen aus der Steigung, dass $y_3 = 1$.

Wir finden dann

$$3P = (2, 3) + (0, 1)$$

erhalten

$$M = 1$$

$$x_3 = -1$$

$$y_3 = 0$$

also

$$3P = (2, 3) + (0, 1) = (-1, 0)$$

Wir finden dann

$$4P = (2, 3) + (-1, 0)$$

erhalten

$$M = 1$$

$$x_3 = 0$$

$$y_3 = -1$$

also

$$4P = (2, 3) + (-1, 0) = (0, -1)$$

Wir finden dann

$$5P = (2, 3) + (0, -1)$$

erhalten

$$M = 2$$

$$x_3 = 2$$

$$y_3 = -3$$

also

$$5P = (2, 3) + (-1, 0) = (2, -3)$$

Jetzt, wenn wir versuchen zu addieren, um

$$6P = (2, 3) + (2, -3)$$

zu bekommen, erhalten wir eine Null im Nenner für M . Die beiden Punkte liegen auf einer vertikalen Linie, sie sind also invers zueinander in der Gruppe, und der dritte Punkt auf der Linie ist der Punkt im Unendlichen. Was daran etwas niedlich ist, ist, dass wir die Antwort, die wir wollen, genau an dem Punkt finden, an dem die Arithmetik versagt.

7.2 Beobachtung

Man sollte beachten, dass, da wir

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2$$

haben, die Nenner in den Brüchen im Wesentlichen mit jeder Hinzufügung von Punkten quadriert werden. Daher ist die Arithmetik, als rationale Zahlenarithmetik, ziemlich schrecklich; mit jeder Addition oder Verdoppelung verdoppelt sich im Wesentlichen die Anzahl der Bits, die benötigt werden, um die Koordinaten eines Punktes darzustellen.

7.3 Projektive Koordinaten und Jacobische Koordinaten

In vielen Fällen ist es einfacher, zu *projektiven Koordinaten* oder *jacobischen Koordinaten* X, Y, Z anstelle von Standardkoordinaten x, y zu wechseln. Das ist eigentlich ganz einfach: Wir entscheiden uns dafür, nicht mit

$$\mathcal{E} : y^2 = x^3 + ax + b$$

zu arbeiten, weil es uns zwingen würde, rationale Arithmetik zu betreiben. Stattdessen stellen wir fest, dass, wenn a und b Ganzzahlen sind, dann die rationalen Zahlen, die die Koordinaten sind

$$(x, y)$$

werden

$$(X/Z^2, Y/Z^3)$$

für Ganzzahlen X, Y , und Z . Wenn wir die Nenner eliminieren, erhalten wir die Gleichung für die Kurve als

$$\mathcal{E} : Y^2 = X^3 + aXZ^4 + bZ^6$$

Wir betrachten einen Punkt auf der Kurve als das Dreifache

$$(X : Y : Z)$$

alle davon sind Ganzzahlen.

Was dies wirklich tut, nachdem man alle Formeln für das Verdoppeln oder Hinzufügen von Punkten neu geschrieben hat, ist eine sanftere Methode (sanfter, das heißt, als Division durch Null) zu bieten, um festzustellen, dass man dabei ist, zwei Punkte hinzuzufügen, die auf einer vertikalen Linie liegen. Wir werden einen algorithmischen Prozess für die Punktaddition und -verdoppelung in Kap. 14 vorstellen.

7.4 Ein Beispiel für eine Kurve mit vielen Punkten

Eine nichttriviale Kurve mit überraschend vielen Ganzzahlpunkten ist

$$\mathcal{E} : y^2 = x^3 - 3024x - 1353456$$

die eine große Anzahl von Punkten mit Ganzzahlkoordinaten aus Tab. 7.1 hat.

Und wir finden eine sehr große Anzahl von Kollineationen von Punkten, alle mit ganzzahligen Koordinaten. Das ist nicht üblich. Wir bemerken die Muster in den Kollineationen der Tab. 7.2: Wenn alle Indizes ungerade sind, ist die Kollineation, die sich aus der Negation aller y -Koordinaten ergibt, die gleiche wie das Hinzufügen von eins zu jedem der Indizes. Ebenso entspricht die Negation der y -Koordinaten der Kollineation

$$P_1 + P_4 + P_{20} = \mathcal{O}$$

der Kollineation

Tab. 7.1 Punkte mit Ganzzahlkoordinaten auf $\mathcal{E} : y^2 = x^3 - 3024x - 1353456$

P_1	$= (120, 108)$	P_2	$= (120, -108)$
P_3	$= (156, 1404)$	P_4	$= (156, -1404)$
P_5	$= (192, 2268)$	P_6	$= (192, -2268)$
P_7	$= (228, 3132)$	P_8	$= (228, -3132)$
P_9	$= (436, 8956)$	P_{10}	$= (436, -8956)$
P_{11}	$= (552, 12852)$	P_{12}	$= (552, -12852)$
P_{13}	$= (588, 14148)$	P_{14}	$= (588, -14148)$
P_{15}	$= (777, 21573)$	P_{16}	$= (777, -21573)$
P_{17}	$= (1020, 32508)$	P_{18}	$= (1020, -32508)$
P_{19}	$= (1488, 57348)$	P_{20}	$= (1488, -57348)$
P_{21}	$= (3585, 214623)$	P_{22}	$= (3585, -214623)$
P_{23}	$= (10056, 1008396)$	P_{24}	$= (10056, -1008396)$
P_{25}	$= (22080, 3280932)$	P_{26}	$= (22080, -3280932)$
P_{27}	$= (34356, 6368004)$	P_{28}	$= (34356, -6368004)$
P_{29}	$= (561360, 420593148)$	P_{30}	$= (561360, -420593148)$

Tab. 7.2 Punktindizes für Kollineationen

(1 3 17)	(1 4 20)	(1 5 13)	(1 6 16)
(1 7 9)	(1 8 12)	(2 3 19)	(2 4 18)
(2 5 15)	(2 6 14)	(2 7 11)	(2 8 10)
(3 5 7)	(3 6 24)	(3 8 22)	(3 10 16)
(3 12 14)	(4 5 23)	(4 6 8)	(4 7 21)
(4 9 15)	(4 11 13)	(5 8 26)	(5 10 20)
(5 12 18)	(6 7 25)	(6 9 19)	(6 11 17)
(7 14 20)	(7 16 18)	(8 13 19)	(8 15 17)
(9 12 28)	(9 14 26)	(9 18 22)	(10 11 27)
(10 13 25)	(10 17 21)	(11 14 30)	(11 16 26)
(11 20 22)	(12 13 29)	(12 15 25)	(12 19 21)
(13 16 28)	(13 18 24)	(14 15 27)	(14 17 23)
(15 20 24)	(16 19 23)	(17 20 28)	(18 19 27)
(21 24 26)	(22 23 25)	(25 28 30)	(26 27 29)

$$P_2 + P_3 + P_{19} = \mathcal{O}$$

wo die y -Koordinaten ähnlich negiert wurden.

7.5 Kurven Modulo einer Primzahl p

Was wir oben diskutiert haben, sind Kurven, deren Punkte aus dem Feld der rationalen Zahlen genommen werden.

Wir können ebenso gut Kurven betrachten, deren Punkte aus einem Feld von Ganzzahlen modulo einer Primzahl p genommen werden, oder aus einem Feld, das durch ein irreduzibles Polynom mit Koeffizienten modulo 2 erzeugt wird.

Zum Beispiel, wenn wir Punkte modulo 11 auf der Kurve

$$\mathcal{E} : y^2 = x^3 + x + 6$$

betrachten, dann sind die Punkte

$$\begin{aligned} &(2, 4), (2, 7) \\ &(3, 5), (3, 6) \\ &(5, 2), (5, 9) \\ &(7, 2), (7, 9) \\ &(8, 3), (8, 8) \\ &(10, 2), (10, 9) \end{aligned}$$

und wenn wir die Kurvenarithmetik durchführen, aber diesmal alle Arithmetik modulo 11 durchführen, erhalten wir

$$\begin{array}{ll}
 P = (2, 4) & 7P = (7, 9) \\
 2P = (5, 9) & 8P = (3, 6) \\
 3P = (8, 8) & 9P = (10, 2) \\
 4P = (10, 9) & 10P = (8, 3) \\
 5P = (3, 5) & 11P = (5, 2) \\
 6P = (7, 2) & 12P = (2, 7) \\
 13P & = \mathcal{O}
 \end{array}$$

Wir beobachten, was wahr sein muss: Da dies eine zyklische Gruppe mit 13 Elementen ist, dann wenn $kP = (x, y)$, haben wir $(13 - k)P = (x, -y)$, weil das additive Inverse eines Punktes der Punkt mit der gleichen x -Koordinate, aber dem Negativen der y -Koordinate ist.

7.6 Hasse's Theorem

Wir werden später in Abschn. 11.2.1 und darüber hinaus sehen, dass Kryptographie in endlichen Gruppen modulo einer großen Primzahl p durchgeführt werden kann, vorausgesetzt, dass die Ordnung der Gruppe etwa die gleiche Größe wie p hat und dass die Identität der Gruppe mit einem algebraischen Ausdruck verbunden werden kann, der leicht modulo p erkannt werden kann. Wir haben mit der projektiven Darstellung der Kurve gesehen, dass die zweite Bedingung erfüllt ist: Die Identität der Kurvengruppe kann erkannt werden, wenn die z -Koordinate modulo p null wird.

Dass die erste Bedingung, bezüglich der Größe der Gruppe, erfüllt ist, ist Hasse's Theorem [1, S. 82] .

Satz 7.1 (Hasse's Theorem) Sei $\#\mathcal{E}(F_p)$ die Anzahl der Punkte auf einer elliptischen Kurve \mathcal{E} genommen modulo einer Primzahl p . Dann gilt

$$\#\mathcal{E}(F_p) = p + 1 - t$$

wo wir haben

$$|t| < 2\sqrt{p}$$

Das heißt, die Anzahl der Punkte auf einer Kurve modulo p liegt innerhalb von $2\sqrt{p}$ von $p + 1$.

Dies wird für kryptographische Zwecke wichtig, weil es besagt, dass die Größe der Gruppe, $\mathcal{O}(p)$, groß genug ist, um eine Gruppe auf der Kurve zu gewährleisten, die

rechnerisch nicht vollständig untersucht werden kann. Es wird nicht der Fall sein, dass alle Gruppen für den Einsatz in kryptographischen Anwendungen geeignet sind, aber es wird ausreichend viele geben, die wir für kryptographische Zwecke verwenden können.

7.7 Übungen

1. Berechnen Sie für die Kurve von Abschn. 7.4 die Summen

$$2P_1$$

$$2P_3$$

$$P_1 + P_3$$

$$P_2 + P_5$$

und beachten Sie, dass die letzten beiden in den Kollineationen von Tab. 7.2 aufgeführt sind.

2. Gegeben ist eine elliptische Kurve

$$\mathcal{E} : y^2 = x^3 + Ax + B,$$

Zeigen Sie, dass die rationalen Punkte (x, y) auf \mathcal{E} die Form $(a/e^2, b/e^3)$ haben, wobei a, b und e ganze Zahlen sind und die Brüche in niedrigsten Termen ausgedrückt sind.

3. Berechnen Sie die Punkte modulo 11 und 13 auf der Kurve

$$y^2 = x^3 - 6x + 9$$

Berechnen Sie die Gruppe modulo 11.

4. (Programmierübung.) Schreiben Sie ein Programm, um die Anzahl der Punkte auf einer Kurve modulo einer Primzahl p zu zählen. Für kleine Primzahlen ist dies nicht schwierig: Führen Sie die Schleife auf mögliche Werte x aus, berechnen Sie die rechte Seite $RHS = x^2 + Ax + B$ modulo p , und bestimmen Sie dann mit Hilfe des quadratischen Restsymbols, ob RHS ein Quadrat modulo p ist. Wenn dies der Fall ist, trägt dieses x einen oder zwei Punkte zur Zählung bei, abhängig davon, ob $RHS \neq 0$ ist oder nicht.
5. (Programmierübung.) Schreiben Sie den Code, um Arithmetik auf einer elliptischen Kurve modulo einer Primzahl p durchzuführen. Testen Sie Ihren Code auf Kurven modulo der Primzahl 257, indem Sie Punkte auf der Kurve finden und dann die Ordnungen dieser Punkte durch das additive Analogon der „Exponentiation“ finden. Sie sollten in der Lage sein zu erkennen, dass Ihr Code funktioniert, indem Sie überprüfen, dass für jeden Punkt P ein n existiert, so dass nP die Identität ist. Sie möchten vielleicht die anfängliche Berechnung mit Jacobischen Koordinaten durchführen, aber dann die Punkte auf Punkte mit $z = 1$ reduzieren, weil das Ihnen leichter erlaubt, Punkte und ihre Inversen im Zyklus zu erkennen.

6. (Wahrscheinliche Programmierübung.) Viele der Kurvengruppen, die wir als Beispiele verwendet haben, sind einzelne Zyklen. Berechnen Sie die Gruppe für $\mathcal{E} : y^2 = x^3 + x + 18$ modulo 31, um zu zeigen, dass die Gruppe kein einzelner Zyklus ist. (Hinweis: Betrachten Sie die Punkte (1, 12) und (2, 11)).

Literatur

1. D. Hankerson, A. Menezes, S. Vanstone, *Guide to elliptic curve cryptography* (Springer, Berlin, 2004)



Zusammenfassung

Wir haben früher bemerkt, dass die tatsächliche Durchführung von Kryptographie das Kombinieren von Mathematik und Informatik erfordert. In diesem Kapitel beschreiben wir mehrere Algorithmen und Rechentricks, die es ermöglichen, die diskrete Mathematik, die Kryptographie ist, auf Computern durchzuführen, die nicht unbedingt darauf ausgelegt sind, robuste Unterstützung für diskrete Mathematik zu bieten. Dieses Kapitel behandelt einige dieser Tricks und Algorithmen, die notwendig sind, um zu verstehen, wie man tatsächlich Kryptographie in der realen Welt durchführen könnte. Die erste Reihe von Tricks wurde ausgiebig beim Testen von Ganzzahlen auf Primzahleigenschaft verwendet, wobei die Bitmuster der Ganzzahlen verwendet wurden, um die Notwendigkeit einer Modulreduktion zu eliminieren. Multipräzise Arithmetik ist für einen Großteil der modernen Kryptographie erforderlich, wobei die Modulreduktion und Multiplikation die Kosten der Arithmetik dominieren. Die Multiplikation selbst wird mit schnellen Methoden wie der FFT durchgeführt, die wir hier behandeln, und die Reduktion kann mit der Montgomery-Multiplikation behandelt werden, die im Wesentlichen den Mersenne-Primzahltrick auf alle Ganzzahlmoduli erweitert.

8.1 Mersenne Primzahlen

In regelmäßigen Abständen (zum Beispiel am 7. Dezember 2018) wird bekannt gegeben, dass eine neue Mersenne Primzahl entdeckt wurde. Dies ist eine kurze Einführung, wie und warum das passiert, warum die größten bekannten Primzahlen fast immer Mersenne-Zahlen sind und warum diese scheinbar esoterische Aktivität in gewisser Weise ein Modell für die reale Datenverarbeitung ist. Die zu lernende Lektion ist das

Zusammenspiel zwischen Mathematik und Datenverarbeitung. Man kann beweisen, dass das Testen bestimmter Zahlen auf Primzahleigenschaft schneller durchgeführt werden kann (theoretisch, mit einer großen \mathcal{O} -Schätzung) als das Testen einer zufälligen Zahl, und glücklicherweise gibt es für diese bestimmten Zahlen einen Rechentrick, der das Testen auf Primzahleigenschaft nicht nur theoretisch machbar, sondern tatsächlich praktikabel macht in Praxis.

8.1.1 Einführung

Die größte bekannte Primzahl zum Zeitpunkt dieser Schrift ist

$$2^{82589933} - 1,$$

eine Zahl mit 24, 862, 048 Dezimalstellen, gefunden am 7. Dezember 2018.

Dies ist die siebte Zahl mit mehr als zehn Millionen Dezimalstellen, die als Primzahl bewiesen wurde. (Eine Zahl der Form $2^N - 1$ wird als *Mersenne-Zahl* bezeichnet.)

Obwohl dies wie ein eher abstraktes Unterfangen erscheinen mag, ist der Nachweis solcher Zahlen als Primzahlen eine Kombination aus Theorie, Algorithmus und Implementierung, die als gutes Modell für das Lösen von Problemen dient, insbesondere von Problemen in der diskreten Mathematik, auf einem Computer.

Es stellt sich auch heraus, dass, wenn man die Rechenkosten für die Suche nach der größten bekannten Primzahl analysiert, diese Kosten ziemlich gut mit der rohen Rechenleistung der besten verfügbaren Rechenplattformen übereinstimmen. Im Wesentlichen kann man argumentieren, dass die kleine Gruppe von extremen Mersenne-Fans uns einen Gefallen tut, indem sie eine Berechnung durchführt, die eine sonst schwierige Messung der rohen Leistungsfähigkeit der Datenverarbeitung nachvollzieht.

8.1.2 Theorie

8.1.2.1 Allgemeine Theorie

Zunächst einmal, wie beweisen wir, dass eine Zahl eine Primzahl ist? Es gibt einen allgemeinen Algorithmus namens AKS Algorithmus (nach den Initialen der Erfinder), der recht schnell läuft [1]. Dies sorgte im Jahr 2002 für Aufsehen, da es der erste Algorithmus zum Testen der Primzahleigenschaft von Ganzzahlen war, der nachweislich in polynomialer Zeit läuft. Im Falle des Primzahlnachweises bedeutet dies polynomial in der Anzahl der Bits in der zu testenden Zahl. Wenn die Zahl P ist und N Bits hat (und wir somit $2^{N-1} \leq P < 2^N$ haben), dann läuft AKS in einer Zeit, die polynomial in N ist, was polynomial in $\lg P$ ist. Tatsächlich läuft die aktuell beste Version in einer Zeit von $\mathcal{O}((\lg P)^6)$.

So gut dieser Algorithmus auch ist, er ist jedoch zu kompliziert, um eine Zahl von fast 25 Millionen Dezimalstellen zu testen. Stattdessen müssen wir uns auf einfachere Methoden verlassen, wenn wir diese riesigen Zahlen testen wollen.

Die Grundlage für diese Art von heroischer Berechnung ist der kleine Satz von Fermat.

8.1.2.2 Allgemeinere Theorien

Wir wiederholen Fermats kleinen Satz aus Kap. 3.

Satz 8.1 (Fermats (kleiner) Satz) Für jede Primzahl P , und jede ganze Zahl a , dann teilt P $a^{P-1} - 1$.

Tatsächlich, wie bereits erwähnt, ist dies wirklich nur Lagranges Satz dass jedes Element in einer Gruppe, erhöht zur Ordnung der Gruppe, ist die Identität. Die Ordnung der multiplikativen Gruppe modulo einer Primzahl P ist gerade $P - 1$, also ist dies nur Lagrange.

Dies ist ein sehr nützlicher Satz, aber es ist kein Satz, der verwendet werden kann, um zu beweisen, dass eine Zahl prim ist. Was dies besagt, ist, dass wenn für jede a wir haben, dass a^{P-1} ist *nicht* 1 modulo P , dann ist P *nicht* eine Primzahl. Aber für allgemeine Zahlen funktioniert es nur in diese Richtung und ist kein „wenn und nur wenn“ Satz. Tatsächlich gibt es eine unendliche Menge von sogenannten *Carmichael-Zahlen* welche sind ganze Zahlen C , die *nicht* prim sind, aber für welche N teilt $a^{C-1} - 1$ für alle a . Die kleinste Carmichael-Zahl ist 561.

8.1.2.3 Spezielle Zwecktheorie

Einer der Gründe, warum die größten bekannten Primzahlen (fast) immer Mersenne-Primzahlen sind, ist, dass es für diese spezielle Art von Zahl tatsächlich möglich ist, eine wenn-und-nur-wenn-Variation des kleinen Fermatschen Satzes zu verwenden.

Satz 8.2 ((Lucas-Lehmer-Test) [2, S. 223–225]) Sei N eine ungerade Primzahl, die kongruent zu 3 modulo 4 ist. Lassen Sie $P = 2^N - 1$ eine Mersenne-Zahl sein, definieren Sie $s_1 = 3$ und berechnen Sie rekursiv

$$s_i = s_{i-1}^2 - 2 \pmod{P}$$

für $i > 1$. Wenn P s_{N-1} teilt, das heißt, wenn der Rest s_{N-1} null modulo P ist, dann ist P eine Primzahl. Andernfalls ist P keine Primzahl.

Dies ist nicht wirklich nur Lagrange (oder kleiner Fermat), aber es ist ziemlich ähnlich, und wir können die Primzahleigenschaft sowohl ja als auch nein aufgrund des Ergebnisses dieses Tests bestimmen.

Dies ist ein Test, der eine Anzahl von Rechenschritten erfordert, die logarithmisch zur Größe von $2^N - 1$ ist (das heißt, linear in N – wir benötigen $\mathcal{O}(N)$ Quadrierungen). Dies

ist asymptotisch nicht viel anders als der allgemeine AKS-Algorithmus, aber natürlich funktioniert dies nur für Mersenne-Zahlen.

Die Bedingung, dass N eine ungerade Primzahl ist, ist keine wichtige Bedingung, denn die Algebra der Oberstufe zeigt, dass

$$2^{ab} - 1 = (2^a - 1) \cdot (2^{ab-a} + 2^{ab-2a} - 2^{ab-3a} \dots + 1)$$

es also eine algebraische Faktorisierung von $2^{ab} - 1$ gibt, wenn sowohl a als auch b größer als 1 sind und wir nicht einmal einen echten Primzahltest durchführen müssten.

8.1.2.4 Vorverarbeitung

In jeder realen Suche nach Primzahlen möchten wir so schnell wie möglich die Ganzzahlen herausfiltern, die unmöglich Primzahlen sein können. Zum Beispiel, wenn $N \equiv 4 \pmod{5}$, dann wird $2^N - 1$ null modulo 5 sein. Es gibt eine ähnliche Reihe von Bedingungen, die für alle Primzahlen angewendet werden können, so dass eine Suche nach Mersenne-Primzahlen damit beginnt, alle Exponenten herauszufiltern, die unmöglich funktionieren können. Im Allgemeinen lautet das Mantra, dass man so lange filtert, bis man es einfach nicht mehr ertragen kann, oder bis der Kostennutzen des Laufens des Filters unter die Kosten des Testens eines Exponenten fällt, der nach dem Filter übrig bleibt.

8.1.2.5 Algorithmen

Angenommen, wir haben einen Satz, der angeblich in polynomialer Zeit in der Anzahl der Bits läuft, wie können wir dies rechnerisch effektiv machen?

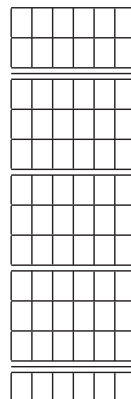
Tatsache Wir müssen s_{N-2} nicht in voller Genauigkeit berechnen; wir müssen nur s_{N-2} modulo P berechnen, so dass wir bei jedem Quadrieren modulo P reduzieren können. Unsere Arithmetik wird nur doppelt so groß wie P selbst. Das heißt, wir müssen nur mit Ganzzahlen der Größe $2N$ Bits umgehen. \square

Dies ist eine offensichtliche Tatsache für einen Zahlentheoretiker, aber es ist auch sehr mächtig in der Berechnung, weil es die Größe der Zahlen klein hält. Wenn a eine Zahl von n Bits ist, dann ist a^2 eine Zahl von $2n$ Bits (plus oder minus eins oder zwei, je nachdem, ob Überträge vorliegen oder nicht), und a^4 ist eine Zahl von $4n$ Bits. Die tatsächliche Zahl, die ohne Reduktion s_{N-2} wäre, wäre eine Zahl von $2^{N-2} \approx P$ Bits, exponentiell größer als P selbst.

Aber weil wir modulo P reduzieren dürfen, könnten wir die notwendige Arithmetik auf etwa 25 Millionen Dezimalstellen an jeder Stelle reduzieren, wenn wir das derzeit größte bekannte Primzahl testen würden.

Der Hauptalgorithmus, den wir verwenden, ist die *schnelle Multiplikation mit der Fast Fourier Transformation (FFT)* [3, 4]. Wenn wir nicht zu sehr nachdenken würden, würden wir die Quadrierungen mit naiver Schulkindarithmetik durchführen, die im Grunde genommen darin besteht, ein Array von Zahlen mit einem anderen Array von

Abb. 8.1 Naive Multiplikation von Ganzzahlen



Zahlen zu multiplizieren, die Produkte in einem dritten Array zu sammeln und die Überträge zu regeln (Abb. 8.1).

Auf modernen Computern würden wir dies nicht in Basis 10, sondern in Basis 2^{32} tun, da wir auf einem ernsthaften Computer ein $32 \times 32 = 64$ -Bit-Produkt ohne Überlauf ansammeln können. Für diese größte Mersenne-Primzahl würde eine einzelne Berechnung eines Zwischenwerts von s_i $82589933/32 = 2580936$ „Ziffern“ (aufgerundet für die letzte Ziffer) benötigen. Naiv betrachtet, benötigt die Multiplikation einer n -Ziffern-Zahl mit einer m -Ziffern-Zahl $n \times m$ Multiplikationen. Quadrieren ist doppelt so schnell wie Multiplizieren, weil wir keine unterschiedlichen Kreuzprodukte haben, aber ein naives Quadrieren benötigt immer noch $\mathcal{O}(n^2)$ Multiplikationen.

Die *Schnelle Fourier-Transformation*, oder FFT, kann jedoch von ihrer üblichen Verwendung in der Signalverarbeitung angepasst werden, um einen Multiplikationsalgorithmus zu erzeugen, der nicht in $\mathcal{O}(n^2)$ Zeit, sondern in der viel schnelleren $\mathcal{O}(n \log n)$ Zeit läuft.

Algorithmisch gesehen erfordert der Nachweis, dass eine bestimmte Mersenne-Zahl $2^N - 1$ prim ist, etwa N Quadrierungen (und dann eine Subtraktion einer Ziffer) einer Zahl, die N Bits lang ist, und jede dieser Quadrierungen kann in etwa $(N/32) \cdot \log(N/32)$ Ganzzahlmultiplikationen von Einzelpräzisions-32-Bit-Zahlen durchgeführt werden. Nach jeder Quadrierung müssen wir ein doppelt so langes Produkt von $2N$ Bits auf ein Ergebnis von N Bits reduzieren.

8.1.3 Implementierung

Wir sind immer noch nicht in der Lage, die Primzahlprüfung in machbarer Zeit nur mit diesen Vereinfachungen durchzuführen. Wenn die Komplexität der FFT-Multiplikation genau $n \log n$ wäre, dann würde eine einzige Quadratur einer Zahl von 82589933 Bits etwa 55 Millionen Ganzzahl-Multiplikationsmaschinenanweisungen benötigen, und die

Gesamtzahl der Multiplikationen wäre mehr als 10^{15} . Bei 10 ns pro Multiplikation liegen die Kosten für die Multiplikation in der Größenordnung von 10^7 s, das sind etwa vier Monate. Dies wäre akzeptabel, wenn dies alles wäre, was wir tun müssen, und wenn wir nur einen einzigen Exponenten testen müssten, aber tatsächlich haben wir dieses unangenehme Geschäft, ein doppelt langes Produkt auf einfache Länge zu reduzieren. Leider ist eine echte mehrfache Division normalerweise vielleicht 50–100 Mal teurer als eine Multiplikation. Wenn wir eine echte Division durch $2^N - 1$ durchführen müssten, um den doppelt langen quadrierten Wert auf eine einfache Länge für die nächste Quadratur zu reduzieren, könnten wir den Test einer Mersenne-Zahl auf Primzahligkeit nicht abschließen.

An dieser Stelle verlassen wir uns auf einen Implementierungstrick, der auf der Natur von Computern basiert, die mit binärer Arithmetik gebaut sind. Jeder der Werte s_i , die in den Zwischenschritten des Lucas-Lehmer-Tests berechnet werden, können als eine einzelne „Ziffer“ D Basis 2^N geschrieben werden. Wenn wir D quadrieren, erhalten wir eine „zweistellige“ Zahl $A \cdot 2^N + B$, wobei jede von A und B einzelne Ziffern Basis 2^N sind, das heißt, sie sind Ganzzahlen von N Bits Länge. Wir führen jetzt ein kleines Stück algebraischer Taschenspielertricks durch:

$$\begin{aligned} A \cdot 2^N + B &= \\ A \cdot 2^N - A + A + B &= \\ A \cdot (2^N - 1) + A + B &\equiv \\ \equiv A + B \pmod{2^N - 1}. \end{aligned}$$

Wenn wir daran interessiert sind, das Produkt modulo $2^N - 1$ zu reduzieren, dann ist die Aufgabe einfach. Der Rest dieses Ausdrucks bei der Division durch $2^N - 1$ ist eindeutig $A + B$, mit möglicherweise einer Subtraktion durch $2^N - 1$, wenn die Addition einen Übertrag erzeugt hat.

Denken Sie einen Moment darüber nach, was das bedeutet. Im stationären Zustand haben wir einen Zwischenwert s_i , der ein Rest modulo $2^N - 1$ ist, also eine Ganzzahl von N Bits. Wenn wir diese Zahl im Rahmen des Prozesses zur Berechnung von $s_{i+1} = s_i^2 - 2$ modulo $2^N - 1$ quadrieren, erhalten wir eine Ganzzahl von $2N$ Bits, oder von zwei Ziffern Basis 2^N . Um den Rest modulo $2^N - 1$ zu berechnen, müssen wir jedoch überhaupt keine Divisionen durchführen. Stattdessen nehmen wir die linke „Ziffer“ A , das heißt, die linke Hälfte des Produkts, verschieben sie und addieren sie zur rechten Hälfte des Produkts, der rechten „Ziffer“ B . Addition ist eine lineare Zeitoperation, keine quadratische Zeitoperation (wie Multiplikation), und sicherlich nicht so teuer wie die Division selbst.

Wir stellen fest, dass wir beim Zusammenzählen von A und B im Durchschnitt zur Hälfte der Zeit einen Übertrag haben und eine Zahl von $N + 1$ Bits erhalten, die größer ist als $2^N - 1$. Um die Reduktion abzuschließen, müssten wir dann $2^N - 1$ subtrahieren, aber wir müssten nur einmal subtrahieren. Die Kosten der „Division“, die eigentlich eine modulare Reduktion ist, betragen daher für Mersenne-Zahlen nur $3/2$ mal die linearen

Kosten einer einzigen Addition. (Eine Addition die ganze Zeit ist ein linearer Kostenfaktor, plus ein weiterer linearer Kostenfaktor die Hälfte der Zeit im Durchschnitt für die Fälle, in denen die Addition einen Übertrag erzeugt, ergibt im Durchschnitt $3/2$ lineare Operationen.)

8.1.4 Zusammenfassung: Machbarkeit

Die Fähigkeit, riesige Mersenne-Zahlen als Primzahlen zu beweisen, ergibt sich aus der Konvergenz mehrerer Faktoren.

1. Erstens gibt es für diese Zahlen ein theoretisches Ergebnis (der Lucas-Lehmer-Test) das ermöglicht es, mit einem Test zu bestimmen, ob eine Zahl dieser speziellen Form entweder eine Primzahl ist oder keine Primzahl ist.
2. Zweitens gibt es einen grundlegenden Algorithmus (die FFT), der eine naiv n^2 Anzahl von „Schritten“ auf eine handhabbare $n \log n$ Anzahl von „Schritten“ reduziert.
3. Schließlich gibt es einen Implementierungstrick, der eine ansonsten unhandhabbare modulare Division in der innersten Schleife in eine sehr handhabbare lineare Zeitaddition umwandelt.

Ohne das Zusammenwirken all dieser Faktoren wäre es nicht möglich zu beweisen, dass Zahlen dieser Größenordnung Primzahlen sind.

8.1.5 Fermat-Zahlen

Die bisherige Diskussion handelte von Mersenne-Zahlen, in der Form

$$M_n = 2^n - 1.$$

Eine ähnliche Diskussion kann über die Fermat-Zahlen geführt werden

$$F_n = 2^{2^n} + 1.$$

Fermat vermutete, dass diese für alle n prim sind. Tatsächlich wurden sie nur für $n = 0, 1, 2, 3, 4$ als prim beobachtet.

Es gibt einen analogen Test, der auf Pépin zurückgeht, um diese auf Primzahlen zu testen: F_n ist prim, wenn und nur wenn

$$3^{(F_n-1)/2} \equiv -1 \pmod{F_n}.$$

Die gleichen Argumente gelten hier wie für Mersenne-Zahlen. Wir haben einen wenn-und-nur-wenn-Test für Primzahlen, der eine Verfeinerung von Lagranges Theorem in Gruppen ist, und wir können den arithmetischen Trick

$$\begin{aligned}
A \cdot 2^N + B &= \\
A \cdot 2^N + A - A + B &= \\
A \cdot (2^N + 1) - A + B &\equiv \\
&\equiv B - A \pmod{2^N + 1}.
\end{aligned}$$

anwenden, um eine modulare Reduktion nicht durch Division, sondern durch Subtraktion der beiden N -Bit-Hälften eines $2N$ -langen Produkts durchzuführen.

8.1.6 Der arithmetische Trick ist wichtig

Wir werden später über elliptische Kurven Kryptographie sprechen. Die von NIST für die Verwendung in der Kryptographie empfohlenen elliptischen Kurven haben alle Primzahlen, die eine Mersenne/Fermat-ähnliche Form haben als

$$2^n + f(.) + 1$$

wo $f(.)$ ein Polynom ist, das höchstens ein paar Potenzen von 2 ist, so dass analoge schnelle Reduktionsspiele mit der Arithmetik gespielt werden können, indem man Additionen oder Subtraktionen anstelle von Division verwendet.

8.2 Multipräzise Arithmetik und die schnelle Fourier-Transformation

8.2.1 Multipräzise Arithmetik

Viel der modernen Kryptographie basiert auf Arithmetik modulo großen Zahlen, normalerweise entweder Primzahlen oder das Produkt von zwei Primzahlen. Die aktuelle Computerhardware führt normalerweise 32-Bit- oder 64-Bit-Arithmetik durch. Obwohl Python Arithmetik mit Zahlen beliebiger Länge durchführt, haben die meisten Programmiersprachen Ganzzahl-Datentypen mit begrenzter Präzision. Zum Beispiel kann eine `unsigned int_64` Variable in C++ positive Zahlen nicht größer als $2^{64} = 18,446,744,073,709,551,616$ verarbeiten. Allgemeine Arithmetik produziert nicht das korrekte Produkt von zwei Zahlen, es sei denn, sie sind kleiner als $2^{32} = 4,294,967,296$, obwohl eine kleine Zahl mal eine größere Zahl das korrekte Produkt liefert, wenn das Produkt kleiner als 2^{64} ist.

Derzeit verwendet die für die auf Zahlentheorie basierende moderne Kryptographie benötigte Arithmetik Zahlen von mehreren hundert bis vielleicht 4096 Bit Länge. Die zugrunde liegende Software zur Durchführung von Multiplikation, Division, Addition und Subtraktion würde solche Zahlen als Ziffernarrays in einer so großen Basis behandeln, wie es angesichts der Computerhardware und der Programmiersprache machbar wäre; Basis 2^{32} als unsigned Ganzzahlen in C++ wäre vernünftig. Addition und Subtraktion

sind relativ schnell, weil sie lineare Operationen sind; das Addieren von zwei Zahlen mit jeweils d Ziffern erfordert d Additionen, wenn es ziffernweise durchgef hrt wird. Die Multiplikation von zwei Zahlen mit jeweils d Ziffern erfordert jedoch d^2 ziffernweise Multiplikationen, wenn sie mit naiver Sch lermultiplikation durchgef hrt wird.

Die Division ist viel schlimmer, und gl cklicherweise wird die meiste Kryptographie als modulare Arithmetik durchgef hrt und ben tigt keine tats chliche Ganzzahldivision. Was der Divisionsschritt w re, ist die modulare Reduktion, weshalb arithmetische Tricks wie f r Mersenne-Primzahlen verwendet werden, oder wie es als Montgomery-Multiplikation in Abschnitt 8.3 besprochen wird. Im Gro en und Ganzen k nnen wir die tats chliche Division vermeiden, und wir k nnen die naiven d^2 einzelnen Multiplikationsschritte f r eine multipr zise Multiplikation reduzieren, indem wir die Fast Fourier Transform (oder andere Methoden, die algorithmisch schneller sind als Sch leralgorithmen) verwenden.

8.2.2 Hintergrund der FFT

Einer der wichtigsten S tze in der Signalverarbeitung, den wir nicht beweisen oder sogar in einer  berm  ig formalen Weise angeben werden, ist der Fourier-Satz.

Satz 8.3 Jede oszillatorische Kurve kann als Summe von Sinuswellen geschrieben werden.

Das hei t, jede Funktion mit $f(t)$ als Funktion der Zeit t , „im Zeitbereich“ (wie die Ausgabe auf einem Oszilloskop) kann auch „im Frequenzbereich“ als

$$f(\theta) = \sum_{n \in \mathbb{N}} a_n \exp(2\pi i \theta / n)$$

geschrieben werden. Der  bergang von $f(t)$ zur Ermittlung der Koeffizienten a_n f r $f(\theta)$ und zur ck erfolgt durch die *Fourier-Transformation* und ihre Umkehrung.

8.2.3 Polynommultiplikation

Die Polynommultiplikation ist eigentlich dasselbe Problem.

Lassen Sie $f(x) = \sum a_i x^i$ und $g(x) = \sum b_i x^i$

Dann $h(x) = f(x) \cdot g(x) = \sum c_i x^i$ wo

$$c_0 = a_0 b_0$$

$$c_1 = a_0 b_1 + a_1 b_0$$

$$c_2 = a_0 b_2 + a_1 b_1 + a_2 b_0$$

$$c_3 = a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0$$

$$c_4 = a_0 b_4 + a_1 b_3 + a_2 b_2 + a_3 b_1 + a_4 b_0$$

Die Koeffizienten c_i sind *Faltungsprodukte*.

Im Allgemeinen würde es also, um zwei Polynome vom Grad n und m zu multiplizieren, nm Multiplikationen aller Koeffizienten gegen alle anderen Koeffizienten erfordern (und dann einige Additionen, die wir nicht zählen, weil Addition viel billiger ist als Multiplikation).

Aber vielleicht können wir es schneller machen?

1. Bewerten Sie die Polynome f und g an $2n + 1$ Punkten (unter der Annahme, dass $n > m$) Dies kostet Zeit linear in n .
2. Multiplizieren Sie die $2n + 1$ Werte koordinatenweise zusammen. Dies kostet Zeit linear in n .
3. Wir wissen aus der Lagrange-Interpolation (und anderen Ergebnissen), dass es ein einzigartiges Polynom vom Grad $2n$ gibt, das durch $2n + 1$ Punkte geht, und dieses einzigartige Polynom muss daher das Produkt h von f und g sein
4. Interpolieren Sie, um dieses einzigartige h zu bestimmen.

Wenn wir die Lagrange-Interpolation auf naive Weise durchführen, haben wir $2n$ Summanden, von denen jeder $2n$ Multiplikationen erfordert. Dies ist also naiv von der Ordnung n^2 Multiplikationen. Das klingt nach *mehr* Arbeit – wie kann das schneller sein?

Der Trick, der der „schnelle“ Teil der „Schnellen Fourier-Transformation“ ist, besteht darin, dass wenn wir an den richtigen Arten von Punkten interpolieren (in diesem Fall Wurzeln der Einheit, die in einem Moment erklärt werden), dann kollabieren die Summen der Kreuzproduktbegriffe zu Null und müssen nicht berechnet werden.

Und natürlich ist es ein trivialer Sprung von der Polynommultiplikation zur multi-präzisen Multiplikation von Ganzzahlen. Die Dezimalzahl 2345 ist beispielsweise einfach das Polynom

$$2x^3 + 3x^2 + 4x + 5$$

ausgewertet bei der Basis 10. Die Multiplikation von zwei k -stellige Zahlen (in welcher Basis auch immer verwendet wird) erfolgt durch Multiplikation der beiden Polynome von Grad $k - 1$ zusammen und dann Auswertung des Produkt-Polynoms für x gleich der Basis.

8.2.4 Komplexe Zahlen wie benötigt für Fourier-Transformationen

Wir definieren $i = \sqrt{-1}$ und machen einfache Algebra damit, so haben wir

$$i^2 = (\sqrt{-1})^2 = -1,$$

$$i^3 = (\sqrt{-1})^3 = (\sqrt{-1})^2 \cdot (\sqrt{-1}) = -\sqrt{-1} = -i,$$

$$i^4 = (i^2)^2 = (-1)^2 = 1.$$

Betrachten Sie Punkte in der *Argand-Ebene* mit Achsen x und $iy = \sqrt{-1}y$. Wir definieren dann $e^{i\theta}$ als $\cos \theta + i \sin \theta$. Mit dieser Definition ist $e^{i\theta}$ ein Vektor der Lange 1 in der Argand-Ebene, also ein Einheitsvektor mit einem Winkel von θ uber der x -Achse.

Dann $(e^{i\theta})^2 = e^{2i\theta} = \cos 2\theta + i \sin 2\theta$ (machen Sie die Trigonometrie...).

Alles, was wir von der komplexen Arithmetik benotigen, ist, dass die n -ten *Einheitswurzeln*, die Losungen der Gleichung $x^n = 1$, die komplexen Zahlen $e^{2\pi i k/n}$ sind. Dies sind die Punkte auf dem Einheitskreis mit Winkeln $2\pi k/n$, fur $k = 0, 1, \dots, n-1$. Das Multiplizieren dieser Einheitswurzeln kann einfach durch das Addieren der Exponenten erfolgen, was dasselbe ist wie das Addieren der Winkel. Und naturlich konnen Werte von k groer als n modulo n reduziert werden, da wir nur im Kreis herumgehen ...

8.2.5 Die Fourier-Transformation

Wir werden eine 8-Punkt-Schnelle Fourier-Transformation (FFT) als Beispiel verwenden.

Lassen Sie ω eine primitive 8-te Einheitswurzel sein, das heit, $\omega = \exp(2\pi i/8)$, so haben wir $\omega^8 = 1$. Ein Teil des mathematischen Tricks, den wir verwenden werden, um die Faltungsprodukte auf null zusammenfallen zu lassen, ist die Beobachtung, dass

$$0 = (\omega^8 - 1) = (\omega - 1)(\omega^7 + \omega^6 + \omega^5 + \omega^4 + \omega^3 + \omega^2 + \omega^1 + \omega^0).$$

Da wir 0 auf der linken Seite haben, muss einer der beiden Faktoren auf der rechten Seite 0 sein. Es kann nicht $\omega - 1$ sein, weil wir speziell ω zu $e^{2\pi i/8}$ gewahlt haben, was definitiv nicht 1 ist. Daher muss die Summe der Einheitswurzeln im zweiten Term der rechten Seite null sein. Deshalb werden die Kreuzproduktbegriffe in der Fourier-Transformation verschwinden.

Betrachten Sie die Matrix

$$F = (\omega^{ij}) = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^6 & \omega^4 & \omega^2 \\ \omega^0 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & -1 & -\omega^1 & -\omega^2 & -\omega^3 \\ 1 & \omega^2 & -1 & -\omega^2 & 1 & \omega^2 & -1 & -\omega^2 \\ 1 & \omega^3 & -\omega^2 & \omega^1 & -1 & -\omega^3 & \omega^2 & -\omega^1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -\omega^1 & \omega^2 & -\omega^3 & -1 & \omega^1 & -\omega^2 & \omega^3 \\ 1 & -\omega^2 & -1 & \omega^2 & 1 & -\omega^2 & -1 & \omega^2 \\ 1 & -\omega^3 & -\omega^2 & -\omega^1 & -1 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}$$

Wir stellen fest, dass das Inverse $F^{-1} = (\omega^{-ij})$ ist.

Dann, wenn $f(x) = a_0 + a_1x + a_2x^2 + \dots + a_7x^7$ die *diskrete Fourier-Transformation* von f ist

$$\begin{pmatrix} f(\omega^0) \\ f(\omega^1) \\ f(\omega^2) \\ f(\omega^3) \\ f(\omega^4) \\ f(\omega^5) \\ f(\omega^6) \\ f(\omega^7) \end{pmatrix} = F \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

Die Fourier-Transformation führt Koeffizienten zu Punkten. Die inverse Transformation führt Punkte zu Koeffizienten. Aber die Matrixmultiplikation ist eine n^2 Berechnung, weil es n Zeilen sind, die jeweils mit dem Spaltenvektor der Höhe n punktiert sind.

8.2.6 Die Cooley-Tukey Schnelle Fourier-Transformation

Jetzt zum *schnellen* Teil der FFT; wir müssen die n^2 Multiplikationen der Matrixmultiplikation nicht wirklich durchführen. Wir können die Struktur der Einheitswurzeln nutzen, um die Fourier-Transformation in $\mathcal{O}(n \lg n)$ Zeit durchzuführen.

Um ein Polynom $f(x)$ vom Grad n an den n -ten Einheitswurzeln zu bewerten, lassen Sie

$$f^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots a_{n-2}x^{n/2-1}$$

$$f^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots a_{n-1}x^{n/2-1}$$

Das erste Polynom hat gerade Indexkoeffizienten und das zweite hat ungerade Indexkoeffizienten, so dass wir haben

$$f(x) = f^{[0]}(x) + xf^{[1]}(x),$$

und um $f(x)$ an den n -ten Einheitswurzeln zu bewerten, mussen wir $f^{[0]}(x)$ und $f^{[1]}(x)$ bewerten bei

$$(\omega^0)^2, (\omega^1)^2, (\omega^2)^2, \dots, (\omega^{n-1})^2$$

und dann erhalten wir in n Multiplikationen $f(x)$ an allen n -ten Wurzeln.

Satz 8.4 Wenn n positiv und gerade ist, sind die Quadrate der n -ten Einheitswurzeln identisch mit den $n/2$ -ten Einheitswurzeln.

Wir wenden diese Reduktion rekursiv an: Um die n -te Ordnung FT zu berechnen, mussen wir zwei Polynome an den $n/2$ -ten Einheitswurzeln bewerten und dann n Multiplikationen durchfuhren.

Das heit, um die Fourier-Transformation der n -ten Ordnung zu berechnen, mussen wir zwei Fourier-Transformationen der $n/2$ -ten Ordnung durchfuhren und dann n Multiplikationen durchfuhren.

Rekursiv,

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \\ &= 4T\left(\frac{n}{4}\right) + n + n/2 \\ &= \dots \\ &= n \lg n + n(1 + 1/2 + 1/4 + \dots) \\ &= n \lg n + 2n \\ &= \mathcal{O}(n \lg n) \end{aligned}$$

Wir werden zuerst ein Beispiel durchfuhren und dann den algorithmischen Prozess vorstellen, der die FFT implementiert.

8.2.7 Ein Beispiel

Wir werden multiplizieren

$$f(x) = 1 + 2x + 3x^2$$

und

$$g(x) = 9 + 8x + 7x^2 + 6x^3$$

mit Hilfe einer Fourier-Transformation.

Zunächst führen wir die Multiplikation auf die lange Art durch

$$\begin{array}{r}
 1\ 2\ 3\ 0\ 0\ 0 \\
 \times 9\ 8\ 7\ 6\ 0\ 0 \\
 \hline
 9\ 18\ 27\ 0\ 0\ 0 \\
 + 0\ 8\ 16\ 24\ 0\ 0 \\
 + 0\ 0\ 7\ 14\ 21\ 0 \\
 + 0\ 0\ 0\ 6\ 12\ 18 \\
 \hline
 9\ 26\ 50\ 44\ 33\ 18
 \end{array}$$

oder

$$h(x) = 9 + 26x + 50x^2 + 44x^3 + 33x^4 + 18x^5$$

Bemerkung 8.1 In vielen Anwendungen ist es natürlich, Ganzzahlen und Polynome von links nach rechts anstatt von rechts nach links zu schreiben. Dies resultiert größtenteils aus einer zugrunde liegenden Fragestellung, wie man mit Arrays in Software umgeht. Wenn wir Dinge von links nach rechts schreiben und die Berechnung in einem Array implementieren, dann kann die Erweiterung des Arrays (in diesem Fall zur Aufnahme der Koeffizienten von x^4 und x^5 , oder im Falle der gewöhnlichen Arithmetik vielleicht zur Aufnahme eines zusätzlichen Übertrags am oberen Ende), einfach durch Hinzufügen von Speicherelementen am Ende des Arrays erfolgen. Wenn wir in die entgegengesetzte Richtung arbeiten würden, müssten wir schmerzhaftes Spiel mit Indizes spielen, oder vielleicht das Array um einen Platz nach oben verschieben, um einen Übertrag in die Null-Index-Position im Array einzufügen.

Nun, um diese Multiplikation als eine Potenz-von-zwei-Transformation durchzuführen, müssen wir eine Acht-Punkte-Transformation durchführen, da 8 die kleinste Potenz von 2 ist, die größer ist als der Grad des Produkts (der 5 ist).

Wir führen die Multiplikation durch

$$\begin{pmatrix}
 \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\
 \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\
 \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^0 & \omega^2 & \omega^4 & \omega^6 \\
 \omega^0 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\
 \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 \\
 \omega^0 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\
 \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^6 & \omega^4 & \omega^2 \\
 \omega^0 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1
 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

und die Multiplikation

$$\begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^6 & \omega^4 & \omega^2 \\ \omega^0 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix} \cdot \begin{pmatrix} 9 \\ 8 \\ 7 \\ 6 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

um zwei Spaltenvektoren zu erhalten

$$\begin{pmatrix} 1 \cdot \omega^0 + 2 \cdot \omega^0 + 3 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 \\ 1 \cdot \omega^0 + 2 \cdot \omega^1 + 3 \cdot \omega^2 + 0 \cdot \omega^3 + 0 \cdot \omega^4 + 0 \cdot \omega^5 + 0 \cdot \omega^6 + 0 \cdot \omega^7 \\ 1 \cdot \omega^0 + 2 \cdot \omega^2 + 3 \cdot \omega^4 + 0 \cdot \omega^6 + 0 \cdot \omega^0 + 0 \cdot \omega^2 + 0 \cdot \omega^4 + 0 \cdot \omega^6 \\ 1 \cdot \omega^0 + 2 \cdot \omega^3 + 3 \cdot \omega^6 + 0 \cdot \omega^1 + 0 \cdot \omega^4 + 0 \cdot \omega^7 + 0 \cdot \omega^2 + 0 \cdot \omega^5 \\ 1 \cdot \omega^0 + 2 \cdot \omega^4 + 3 \cdot \omega^0 + 0 \cdot \omega^4 + 0 \cdot \omega^0 + 0 \cdot \omega^4 + 0 \cdot \omega^0 + 0 \cdot \omega^4 \\ 1 \cdot \omega^0 + 2 \cdot \omega^5 + 3 \cdot \omega^2 + 0 \cdot \omega^7 + 0 \cdot \omega^4 + 0 \cdot \omega^1 + 0 \cdot \omega^6 + 0 \cdot \omega^3 \\ 1 \cdot \omega^0 + 2 \cdot \omega^6 + 3 \cdot \omega^4 + 0 \cdot \omega^2 + 0 \cdot \omega^0 + 0 \cdot \omega^6 + 0 \cdot \omega^4 + 0 \cdot \omega^2 \\ 1 \cdot \omega^0 + 2 \cdot \omega^7 + 3 \cdot \omega^6 + 0 \cdot \omega^5 + 0 \cdot \omega^4 + 0 \cdot \omega^3 + 0 \cdot \omega^2 + 0 \cdot \omega^1 \end{pmatrix}$$

und

$$\begin{pmatrix} 9 \cdot \omega^0 + 8 \cdot \omega^0 + 7 \cdot \omega^0 + 6 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 \\ 9 \cdot \omega^0 + 8 \cdot \omega^1 + 7 \cdot \omega^2 + 6 \cdot \omega^3 + 0 \cdot \omega^4 + 0 \cdot \omega^5 + 0 \cdot \omega^6 + 0 \cdot \omega^7 \\ 9 \cdot \omega^0 + 8 \cdot \omega^2 + 7 \cdot \omega^4 + 6 \cdot \omega^6 + 0 \cdot \omega^0 + 0 \cdot \omega^2 + 0 \cdot \omega^4 + 0 \cdot \omega^6 \\ 9 \cdot \omega^0 + 8 \cdot \omega^3 + 7 \cdot \omega^6 + 6 \cdot \omega^1 + 0 \cdot \omega^4 + 0 \cdot \omega^7 + 0 \cdot \omega^2 + 0 \cdot \omega^5 \\ 9 \cdot \omega^0 + 8 \cdot \omega^4 + 7 \cdot \omega^0 + 6 \cdot \omega^4 + 0 \cdot \omega^0 + 0 \cdot \omega^4 + 0 \cdot \omega^0 + 0 \cdot \omega^4 \\ 9 \cdot \omega^0 + 8 \cdot \omega^5 + 7 \cdot \omega^2 + 6 \cdot \omega^7 + 0 \cdot \omega^4 + 0 \cdot \omega^1 + 0 \cdot \omega^6 + 0 \cdot \omega^3 \\ 9 \cdot \omega^0 + 8 \cdot \omega^6 + 7 \cdot \omega^4 + 6 \cdot \omega^2 + 0 \cdot \omega^0 + 0 \cdot \omega^6 + 0 \cdot \omega^4 + 0 \cdot \omega^2 \\ 9 \cdot \omega^0 + 8 \cdot \omega^7 + 7 \cdot \omega^6 + 6 \cdot \omega^5 + 0 \cdot \omega^4 + 0 \cdot \omega^3 + 0 \cdot \omega^2 + 0 \cdot \omega^1 \end{pmatrix}$$

wo wir alle Koeffizienten belassen haben, weil wir sie später alle zusammenfallen lassen werden.

Wir führen die komponentenweise Multiplikation durch und wir wissen, was wir bekommen werden, weil dieses Problem klein genug ist, dass wir die Multiplikation von Hand durchführen können.

$$hvec = \begin{pmatrix} 9 \cdot \omega^0 + 26 \cdot \omega^0 + 50 \cdot \omega^0 + 44 \cdot \omega^0 + 33 \cdot \omega^0 + 18 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 \\ 9 \cdot \omega^0 + 26 \cdot \omega^1 + 50 \cdot \omega^2 + 44 \cdot \omega^3 + 33 \cdot \omega^4 + 18 \cdot \omega^5 + 0 \cdot \omega^6 + 0 \cdot \omega^7 \\ 9 \cdot \omega^0 + 26 \cdot \omega^2 + 50 \cdot \omega^4 + 44 \cdot \omega^6 + 33 \cdot \omega^0 + 18 \cdot \omega^2 + 0 \cdot \omega^4 + 0 \cdot \omega^6 \\ 9 \cdot \omega^0 + 26 \cdot \omega^3 + 50 \cdot \omega^6 + 44 \cdot \omega^1 + 33 \cdot \omega^4 + 18 \cdot \omega^7 + 0 \cdot \omega^2 + 0 \cdot \omega^5 \\ 9 \cdot \omega^0 + 26 \cdot \omega^4 + 50 \cdot \omega^0 + 44 \cdot \omega^4 + 33 \cdot \omega^0 + 18 \cdot \omega^4 + 0 \cdot \omega^0 + 0 \cdot \omega^4 \\ 9 \cdot \omega^0 + 26 \cdot \omega^5 + 50 \cdot \omega^2 + 44 \cdot \omega^7 + 33 \cdot \omega^4 + 18 \cdot \omega^1 + 0 \cdot \omega^6 + 0 \cdot \omega^3 \\ 9 \cdot \omega^0 + 26 \cdot \omega^6 + 50 \cdot \omega^4 + 44 \cdot \omega^2 + 33 \cdot \omega^0 + 18 \cdot \omega^6 + 0 \cdot \omega^4 + 0 \cdot \omega^2 \\ 9 \cdot \omega^0 + 26 \cdot \omega^7 + 50 \cdot \omega^6 + 44 \cdot \omega^5 + 33 \cdot \omega^4 + 18 \cdot \omega^3 + 0 \cdot \omega^2 + 0 \cdot \omega^1 \end{pmatrix}$$

Nun multiplizieren wir diesen Spaltenvektor mit der inversen Matrix für die Fourier-Transformation. Diese Matrix ist

$$F^{-1} = (\omega^{-ij}) = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^6 & \omega^4 & \omega^2 \\ \omega^0 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \end{pmatrix}$$

Lassen Sie uns also die Multiplikation Zeile für Zeile durchführen. Die erste Zeile des Ergebnisses wird das Skalarprodukt der ersten Zeile von F^{-1} mit $hvec$, das heißt, von

$$(\omega^0 \ \omega^0 \ \omega^0 \ \omega^0 \ \omega^0 \ \omega^0 \ \omega^0 \ \omega^0)$$

mit $hvec$. Was wir bekommen, ist einfach die Summe der Zeilen von $hvec$, da $\omega^0 = 1$: Das ist

$$\begin{aligned} & 9 \cdot \omega^0 + 26 \cdot \omega^0 + 50 \cdot \omega^0 + 44 \cdot \omega^0 + 33 \cdot \omega^0 + 18 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 \\ & \quad + \\ & 9 \cdot \omega^0 + 26 \cdot \omega^1 + 50 \cdot \omega^2 + 44 \cdot \omega^3 + 33 \cdot \omega^4 + 18 \cdot \omega^5 + 0 \cdot \omega^6 + 0 \cdot \omega^7 \\ & \quad + \\ & 9 \cdot \omega^0 + 26 \cdot \omega^2 + 50 \cdot \omega^4 + 44 \cdot \omega^6 + 33 \cdot \omega^0 + 18 \cdot \omega^2 + 0 \cdot \omega^4 + 0 \cdot \omega^6 \\ & \quad + \\ & 9 \cdot \omega^0 + 26 \cdot \omega^3 + 50 \cdot \omega^6 + 44 \cdot \omega^1 + 33 \cdot \omega^4 + 18 \cdot \omega^7 + 0 \cdot \omega^2 + 0 \cdot \omega^5 \\ & \quad + \\ & 9 \cdot \omega^0 + 26 \cdot \omega^4 + 50 \cdot \omega^0 + 44 \cdot \omega^4 + 33 \cdot \omega^0 + 18 \cdot \omega^4 + 0 \cdot \omega^0 + 0 \cdot \omega^4 \\ & \quad + \\ & 9 \cdot \omega^0 + 26 \cdot \omega^5 + 50 \cdot \omega^2 + 44 \cdot \omega^7 + 33 \cdot \omega^4 + 18 \cdot \omega^1 + 0 \cdot \omega^6 + 0 \cdot \omega^3 \\ & \quad + \\ & 9 \cdot \omega^0 + 26 \cdot \omega^6 + 50 \cdot \omega^4 + 44 \cdot \omega^2 + 33 \cdot \omega^0 + 18 \cdot \omega^6 + 0 \cdot \omega^4 + 0 \cdot \omega^2 \\ & \quad + \\ & 9 \cdot \omega^0 + 26 \cdot \omega^7 + 50 \cdot \omega^6 + 44 \cdot \omega^5 + 33 \cdot \omega^4 + 18 \cdot \omega^3 + 0 \cdot \omega^2 + 0 \cdot \omega^1 \end{aligned}$$

und jetzt können wir den Vorteil sehen, Dinge nicht früher zusammenzufassen. Wenn wir die erste Spalte des obigen Tableaus addieren, erhalten wir

$$9 \cdot \omega^0 = 9 \cdot 1 = 9$$

8 Mal addiert, das ergibt 72. In jeder anderen Spalte erhalten wir den Koeffizienten mal

$$\omega^0 + \omega^1 + \omega^2 + \omega^3 + \omega^4 + \omega^5 + \omega^6 + \omega^7$$

Aber diese Summe ist null, weil dies Einheitswurzeln sind, also haben wir

$$0 = \omega^8 - 1 = (\omega^1 - 1) \cdot (\omega^0 + \omega^1 + \omega^2 + \omega^3 + \omega^4 + \omega^5 + \omega^6 + \omega^7)$$

Das Produkt ist null, aber da der erste Faktor $(\omega^1 - 1)$ nicht null ist, muss der zweite Faktor null sein.

Also, wenn wir jede Spalte außer der Spalte für den Koeffizienten 9 heruntergehen, erhalten wir einen Koeffizienten (26, 50, 44, 33, 18) mal null. Die erste Zeile des Produkts von F^{-1} und $hvec$ ist also nur die 72.

Jetzt machen wir die zweite Zeile, und das wird alles sein, was wir brauchen, um das Muster zu sehen. Nehmen Sie die zweite Zeile von F^{-1} ,

$$(\omega^0 \ \omega^7 \ \omega^6 \ \omega^5 \ \omega^4 \ \omega^3 \ \omega^2 \ \omega^1),$$

und multiplizieren Sie dies mit $hvec$. Wir erhalten

$$\begin{aligned} &9 \cdot \omega^0 + 26 \cdot \omega^0 + 50 \cdot \omega^0 + 44 \cdot \omega^0 + 33 \cdot \omega^0 + 18 \cdot \omega^0 + 0 \cdot \omega^0 + 0 \cdot \omega^0 \\ &\quad + \\ &9 \cdot \omega^7 + 26 \cdot \omega^0 + 50 \cdot \omega^1 + 44 \cdot \omega^2 + 33 \cdot \omega^3 + 18 \cdot \omega^4 + 0 \cdot \omega^5 + 0 \cdot \omega^6 \\ &\quad + \\ &9 \cdot \omega^6 + 26 \cdot \omega^0 + 50 \cdot \omega^2 + 44 \cdot \omega^4 + 33 \cdot \omega^6 + 18 \cdot \omega^0 + 0 \cdot \omega^2 + 0 \cdot \omega^4 \\ &\quad + \\ &9 \cdot \omega^5 + 26 \cdot \omega^0 + 50 \cdot \omega^3 + 44 \cdot \omega^6 + 33 \cdot \omega^1 + 18 \cdot \omega^4 + 0 \cdot \omega^7 + 0 \cdot \omega^2 \\ &\quad + \\ &9 \cdot \omega^4 + 26 \cdot \omega^0 + 50 \cdot \omega^4 + 44 \cdot \omega^0 + 33 \cdot \omega^4 + 18 \cdot \omega^0 + 0 \cdot \omega^4 + 0 \cdot \omega^0 \\ &\quad + \\ &9 \cdot \omega^3 + 26 \cdot \omega^0 + 50 \cdot \omega^5 + 44 \cdot \omega^2 + 33 \cdot \omega^7 + 18 \cdot \omega^4 + 0 \cdot \omega^1 + 0 \cdot \omega^6 \\ &\quad + \\ &9 \cdot \omega^2 + 26 \cdot \omega^0 + 50 \cdot \omega^6 + 44 \cdot \omega^4 + 33 \cdot \omega^2 + 18 \cdot \omega^0 + 0 \cdot \omega^6 + 0 \cdot \omega^4 \\ &\quad + \\ &9 \cdot \omega^1 + 26 \cdot \omega^0 + 50 \cdot \omega^7 + 44 \cdot \omega^6 + 33 \cdot \omega^5 + 18 \cdot \omega^4 + 0 \cdot \omega^3 + 0 \cdot \omega^2 \end{aligned}$$

Jetzt schauen wir uns das genau an. In der 26er Spalte erhalten wir ω^0 die ganze Zeit, so dass die Summe dieser Spalte $8 \cdot 26 = 228$ ist. In den Spalten 9, 50 und 33 und der ersten 0-Spalte erhalten wir die vorherige Summe

$$\omega^0 + \omega^1 + \omega^2 + \omega^3 + \omega^4 + \omega^5 + \omega^6 + \omega^7,$$

so dass diese Spalten auf null summiert werden. In der 44 und der zweiten 0-Spalte haben wir

$$\begin{aligned}
& \omega^0 + \omega^2 + \omega^4 + \omega^6 + \omega^0 + \omega^2 + \omega^4 + \omega^6 \\
&= 1 + \omega^2 - 1 - \omega^2 + 1 + \omega^2 - 1 - \omega^2 \\
&= 0
\end{aligned}$$

und in der 18er Spalte haben wir

$$\begin{aligned}
& \omega^0 + \omega^4 + \omega^0 + \omega^4 + \omega^0 + \omega^4 + \omega^0 + \omega^4 \\
&= 1 - 1 + 1 - 1 + 1 - 1 + 1 - 1 \\
&= 0.
\end{aligned}$$

Was wir also als endgultige Summe fur den zweiten Zeileneintrag in dem Spaltenvektor fur $h(x)$ erhalten, ist einfach die 228.

Wenn wir uns jetzt die dritte Zeile ansehen, bekommen wir etwas vollig ahnliches wie das, was gerade fur die zweite Zeile passiert ist. Alles wird auf null summiert, auer der 50er Spalte, und dort bekommen wir $8 \cdot 50 = 400$.

Da wir die Fourier-Transformation mit Potenzen von 2 Einheitswurzeln durchfuhren, erhalten wir

$$F^{-1} \cdot hvec = \begin{pmatrix} 8 \cdot 9 \\ 8 \cdot 26 \\ 8 \cdot 50 \\ 8 \cdot 44 \\ 8 \cdot 33 \\ 8 \cdot 18 \\ 8 \cdot 0 \\ 8 \cdot 0 \end{pmatrix}$$

und wenn wir die 8 herausdividieren, erhalten wir die Koeffizienten fur $h(x)$.

8.2.8 Der FFT-Schmetterling

Wie implementieren wir dies nun effizient? Details finden Sie in mehreren Referenzen [3, 4]. Der Schlussel ist das Kommunikationsmuster, das als FFT-Schmetterling bekannt ist, dargestellt in Abb. 8.2. Der Schlussel zum Schmetterling ist erstens, dass die Kommunikation von der ersten Spalte der Knoten zur zweiten an Orte erfolgt, die einen entfernt sind, von der zweiten zur dritten an Orte, die zwei entfernt sind, von der dritten zur vierten an Orte, die vier entfernt sind, und so weiter, je nachdem, wie viele Stufen benotigt werden, und zweitens, dass der Fluss von einer Spalte der Knoten zur nachsten eine einzige Multiplikation und eine einzige Addition der Zwischendaten an jedem jeweiligen Knoten ist. Fur den gezeigten Schmetterling sind die acht Positionen fur eine 8-Punkt-FFT, fur die es $\log_2 8 = 3$ Stufen gibt.

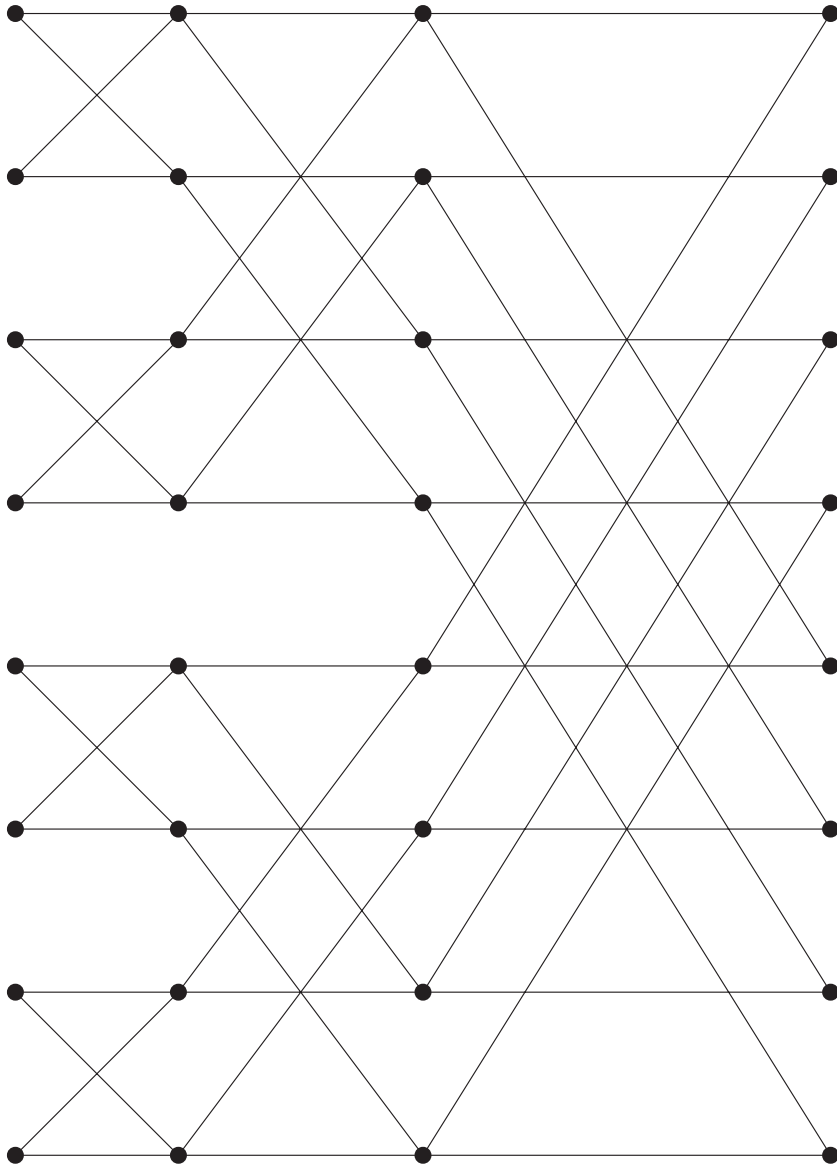


Abb. 8.2 Ein Acht-Punkt FFT-Schmetterling

Die Berechnung selbst ist wie folgt, wobei ω eine achte Einheitswurzel ist. (Die Potenzen von ω die sich in multiplizieren, werden von denen, die FFTs durchführen, „Twiddle-Faktoren“ genannt.) In der ersten Stufe kombinieren wir mit Schritt eins.

a_0	$a_0 + a_4$
a_4	$a_0 + a_4\omega^4$
a_2	$a_2 + a_6$
a_6	$a_2 + a_6\omega^4$
a_1	$a_1 + a_5$
a_5	$a_1 + a_5\omega^4$
a_3	$a_3 + a_7$
a_7	$a_3 + a_7\omega^4$

Dann kombinieren wir bei Schritt zwei.

$a_0 + a_4$	$a_0 + a_4 + a_2 + a_6$
$a_0 + a_4\omega^4$	$a_0 + a_4\omega^4 + (a_2 + a_6\omega^4)\omega^2$
$a_2 + a_6$	$a_0 + a_4 + (a_2 + a_6)\omega^4$
$a_2 + a_6\omega^4$	$a_0 + a_4\omega^4 + (a_2 + a_6\omega^4)\omega^6$
$a_1 + a_5$	$a_1 + a_5 + a_3 + a_7$
$a_1 + a_5\omega^4$	$a_1 + a_5\omega^4 + (a_3 + a_7\omega^4)\omega^2$
$a_3 + a_7$	$a_1 + a_5 + (a_3 + a_7)\omega^4$
$a_3 + a_7\omega^4$	$a_1 + a_5\omega^4 + (a_3 + a_7\omega^4)\omega^6$

Und in der letzten Phase haben wir

$a_0 + a_4 + a_2 + a_6$	$a_0 + a_4 + a_2 + a_6 + a_1 + a_5 + a_3 + a_7$
$a_0 + a_4\omega^4 + a_2\omega^2 + a_6\omega^6$	$a_0 + a_4\omega^4 + a_2\omega^2 + a_6\omega^6 + (a_1 + a_5\omega^4 + a_3\omega^2 + a_7\omega^6)\omega$
$a_0 + a_4 + (a_2 + a_6)\omega^4$	$a_0 + a_4 + a_2\omega^4 + a_6\omega^4 + (a_1 + a_5 + a_3\omega^4 + a_7\omega^4)\omega^2$
$a_0 + a_4\omega^4 + (a_2 + a_6\omega^4)\omega^6$	$a_0 + a_4\omega^4 + a_2\omega^6 + a_6\omega^2 + (a_1 + a_5\omega^4 + a_3\omega^6 + a_7\omega^2)\omega^3$
$a_1 + a_5 + a_3 + a_7$	$a_0 + a_4 + a_2 + a_6 + (a_1 + a_5 + a_3 + a_7)\omega^4$
$a_1 + a_5\omega^4 + (a_3 + a_7\omega^4)\omega^2$	$a_0 + a_4\omega^4 + a_2\omega^2 + a_6\omega^6 + (a_1 + a_5\omega^4 + a_3\omega^2 + a_7\omega^6)\omega^5$
$a_1 + a_5 + (a_3 + a_7)\omega^4$	$a_0 + a_4 + a_2\omega^4 + a_6\omega^4 + (a_1 + a_5 + a_3\omega^4 + a_7\omega^4)\omega^6$
$a_1 + a_5\omega^4 + (a_3 + a_7\omega^4)\omega^6$	$a_0 + a_4\omega^4 + a_2\omega^6 + a_6\omega^2 + (a_1 + a_5\omega^4 + a_3\omega^6 + a_7\omega^2)\omega^7$

Wenn wir die Begriffe der zweiten Spalte dieses Tableaus nach dem Subskriptkoeffizienten umsortieren, erhalten wir das gleiche Muster wie im Beispiel.

Jeder der Knotenpunkte in Abb. 8.2 stellt eine Multiplikation dar (und einige Additionen, um die wir uns nicht kummern, weil Addition viel billiger ist als Multiplikation). Durch die Berechnung mit dem Schmetterlingsmuster haben wir n Multiplikationen in jeder Spalte des Schmetterlings und $\lg n$ Spalten, um die gesamte Berechnung durchzufuhren. Dies ergibt die $\mathcal{O}(n \lg n)$ Laufzeit einer FFT.

8.3 Montgomery-Multiplikation

Die FFT ermöglicht uns eine schnelle Multiplikation. Allerdings besteht ein Großteil der Public-Key-Kryptographie nicht nur aus Multiplikation, sondern aus *modularer* Multiplikation: Bei jedem Schritt des kryptographischen Algorithmus multiplizieren wir zwei Reste modulo N , die $\lg N$ Bits lang sind, um ein Produkt zu erhalten, das $2 \lg N$ Bits lang ist, und dann müssen wir dieses Produkt modulo N reduzieren, um einen Rest im Bereich 0 bis $N - 1$, von $\lg N$ Bits zu erhalten. Diese modulare Reduktion erfordert naiverweise eine Division mit Rest. Die Division mit Rest wird naiverweise durch eine Wiederholung von Subtraktionen und Bitverschiebungen durchgeführt. Für N von, sagen wir, 2048 oder 4096 Bits, könnte das sehr langsam sein.

Die Reduktion modulo N , wie sie für die meisten Public-Key-Kryptographien benötigt wird, kann mit der *Montgomery-Multiplikation*, die von Peter Montgomery erfunden und 1985 veröffentlicht wurde [5], erheblich beschleunigt werden. Die Idee ist so wichtig, dass Hardware für die Durchführung der Montgomery-Multiplikation entwickelt wurde [6, 7].

Die Grundidee ist eine außergewöhnliche Extrapolation aus dem Trick, der für die Arithmetik modulo Mersenne-Zahlen verwendet wird. Hinzu kommt der Trick, der verwendet wird, um die Division selbst rechnerisch schneller zu machen: Wenn wir n durch m teilen wollen, hätten wir normalerweise das Ausprobieren und Korrigieren eines Versuchsquotienten, aber dieses Ausprobieren und Korrigieren kann vereinfacht werden, wenn wir die Operanden vorher mit einer geeigneten Ganzzahl multiplizieren. (Siehe Knuth [8] für die Details.) Die Montgomery-Multiplikation funktioniert auf ähnliche Weise.

Nehmen wir an, wir werden eine Menge Arithmetik modulo einer festen Zahl N durchführen. Wählen Sie $R = 2^k > N$ für ein geeignetes k . Angenommen, R und N sind teilerfremd (und wenn nicht, erhöhen Sie k um eins und wir sollten in der Lage sein, ein R zu bekommen, das teilerfremd ist), dann können wir für R' und N' so lösen, dass $RR' - NN' = 1$.

Was wir dann tun werden, ist alles mit R zu multiplizieren. Alle Konstanten, alle Zahlen usw. werden mit R multipliziert. Anstatt also Arithmetik mit den Ganzzahlen a und b zu betreiben, werden wir Arithmetik mit den Ganzzahlen aR und bR betreiben. Und dann, ganz am Ende der Berechnung, multiplizieren wir jedes Ergebnis mit R' . Da $RR' \equiv 1 \pmod{N}$, erhalten wir das Ergebnis, das wir gehabt hätten.

Addition und Subtraktion sind in Ordnung, da

$$a + b = c \Leftrightarrow aR + bR = cR.$$

Das Problem liegt bei der Multiplikation:

$$aR \cdot bR = abR^2$$

was bedeutet, dass wir einen zusätzlichen Faktor R haben. Was wir tun wollen, ist eine Funktion zu haben, an die wir das Produkt abR^2 übergeben können und die abR zurückgibt. Wir könnten dies tun, indem wir modulo N mit R' multiplizieren, aber das wäre eine Multiplikation modulo N , und genau das versuchen wir zu vermeiden.

So machen wir es. Wir beginnen mit $T = abR^2$.

$$\begin{aligned} m &\leftarrow (T \pmod{R}) \cdot N' \pmod{R} \\ t &\leftarrow (T + mN)/R \end{aligned}$$

und wir geben entweder t oder $t - N$ zurück, je nachdem, welches im Bereich von 0 bis $N - 1$ liegt.

Man könnte denken, dass wir, wenn wir versuchen, die modulare Reduktion zu vermeiden, keinen Gewinn erzielen würden, indem wir die Reduktion durch R zweimal durchführen. Aber ... erinnern Sie sich daran, dass R absichtlich als Potenz von 2 gewählt wurde, und die Reduktion durch eine Potenz von 2 besteht einfach darin, die (ungefähr) rechte Hälfte der Bits des doppelt so großen Produkts auszuwählen und dann die obere Hälfte wegzuworfen.

Beispiel Lassen Sie $N = 79$, und anstelle der Verwendung einer Potenz von 2 für R , verwenden wir $R = 100$ für die Lesbarkeit mit Dezimalzahlen. Wir finden, dass $64 \cdot 100 - 81 \cdot 79 = 1$, also haben wir $R = 100$, $R' = 64$, $N = 79$, $N' = 81$.

Nehmen wir nun an, dass wir $a = 17$ mal $b = 26$ multiplizieren, um 442 zu erhalten. Die Zahl 17 ist wirklich $a' \cdot 100$ modulo 79 für ein bestimmtes a' . Wenn wir $17 \cdot 64 \equiv 61 \pmod{79}$ multiplizieren, stellen wir fest, dass $a' = 61$. Ähnlich ist $26 \cdot 64 \equiv 5 \pmod{79}$. Wenn wir also 17 und 26 in dieser Darstellung multiplizieren, versuchen wir eigentlich, $61 \cdot 5 = 305 \equiv 68 \pmod{79}$ zu multiplizieren.

Da wir wissen, dass wir tatsächlich modulo 79 arbeiten können, wissen wir, dass wir haben

$$\begin{aligned} 17 \cdot 26 = 442 &\equiv (61 \cdot 100) \cdot (5 \cdot 100) \\ &\equiv 305 \cdot 100 \cdot 100 \\ &\equiv 68 \cdot 100 \cdot 100 \pmod{79} \end{aligned}$$

und wenn wir mit 64 multiplizieren und modulo 79 reduzieren, sollten wir die richtige Antwort bekommen:

$$442 \cdot 64 \equiv 28288 \equiv 6 \equiv 68 \cdot 100 \pmod{79}.$$

Die Funktion, die wir wollen, ist die Funktion, die die 442 als Eingabe nimmt und 6 zurückgibt. Und die oben beschriebene Funktion tut genau das:

$$\begin{aligned}
m &= (442 \pmod{100}) \cdot 81 \pmod{100} \\
&= 42 \cdot 81 \pmod{100} \\
&= 3402 \pmod{100} \\
&\equiv 2 \pmod{100} \\
t &= (442 + 2 \cdot 79)/100 \\
&= (442 + 158)/100 \\
&= 600/100 \\
&= 6
\end{aligned}$$

und wir geben $t = 6$ als Ergebnis zurück.

Der Beweis, dass der Algorithmus funktioniert, läuft wie folgt. Wir nehmen an, dass T ein Produkt ist und daher doppelt so lang ist. Da wir $R > N$ wählen, aber nicht viel größer, können die Produkte als doppelt so lang in R genommen werden.

Die erste modulare Reduktion konvertiert einfach T in eine einzelne Länge Nummer modulo R . Wieder modulo R , haben wir, dass $m = TN'$. Daher

$$mN \equiv TN'N \equiv -T \pmod{R}.$$

Wenn wir also $T + mN$ nehmen, erhalten wir eine ganze Zahl, die modulo R null ist und wir können legitim das R herausdividieren und einen ganzzahligen Quotienten für t erhalten.

Jetzt kommt die Tatsache, dass wir den richtigen Quotienten bekommen, von der Tatsache, dass

$$tR = T + mN \equiv T \pmod{N}$$

so dass modulo N wir haben $t \equiv TR'$.

8.3.1 Der Rechenvorteil

Die Montgomery-Multiplikation ersetzt eine mehrfache Division von zwei Ganzzahlen durch eine Reduktion modulo R , eine Multiplikation modulo R zur Erzeugung von m , eine Addition und dann eine Division durch R . Dies klingt zunächst nicht nach einem Gewinn, aber ... wenn R eine Potenz von zwei ist (und nicht die Potenz von 10 des Beispiels), dann ist die Reduktion der rechten Seite, die zur Erzeugung von m modulo R verwendet wird, einfach die rechten Bits dieses Produkts zu nehmen, und dann ist die Division durch R einfach die linken Bits zu nehmen. Es besteht natürlich, wie bei dem Mersenne-Primzahl-Trick, die Möglichkeit, dass unsere Ganzzahlen ein Bit zu lang werden, aber die mögliche eine weitere Subtraktion zur Korrektur ist ein sehr kleiner Preis.

8.4 Allgemeine Arithmetik

Wir bemerken schließlich, dass die Arithmetik auf großen Ganzzahlen entscheidend für die Public-Key-Kryptographie ist und sie, wie auch Berechnungen in der Zahlentheorie im Allgemeinen, daher ausgiebig studiert wurde [9]. Es wurde Hardware entworfen und gebaut [7, 10–15]. Es wurden Software und Algorithmen entwickelt [16–20]. Dies bleibt ein Thema der Untersuchung, insbesondere da die Schlüsselgrößen zunehmen; die theoretisch schnelleren Algorithmen, die für kleinere Schlüsselgrößen möglicherweise nicht praktisch schneller waren, könnten für größere Schlüssel praktisch werden.

8.5 Übungen

1. (Programmierübung.) Überprüfen Sie die Primzahleigenschaft von $M_{13} = 8191$ durch Implementierung des Lucas-Lehmer-Tests. Beachten Sie, dass dies die größte Mersenne-Primzahl ist, für die die Arithmetik mit 32-Bit-Arithmetik ohne Mehrpräzisionsroutinen durchgeführt werden kann.
Wenn Sie Zugang zu einem Computer haben, der 64-Bit-Arithmetik verarbeiten kann, können Sie Ihren Code an M_{17} , M_{19} , und M_{31} testen.
Sie möchten vielleicht zuerst M_{13} ohne den arithmetischen Trick durchführen und dann Ihren Code verbessern, nachdem Sie wissen, dass Sie den Algorithmus richtig implementiert haben. Wenn Sie die größeren Mersenne-Primzahlen bearbeiten können, möchten Sie vielleicht die Unterschiede in den Laufzeiten überprüfen und das Wachstumsmuster bemerken.
2. Machen Sie das Beispiel aus Abschnitt 8.2.7, aber machen Sie es mit dem FFT-Schmetterling aus Abb. 8.2.
3. Machen Sie das Montgomery-Multiplikationsbeispiel von 17 mal 26, aber diesmal machen Sie es modulo $N = 83$.
4. (Programmierübung.) Schreiben Sie ein Programm für die Montgomery-Multiplikation. Testen Sie es mit dem Beispiel und stellen Sie dann sicher, dass es für 123 mal 456 modulo 1009 funktioniert.

Literatur

1. M. Agrawal, N. Kayal, N. Saxena, PRIMES is in P. *Ann. Math.* **160**, 781–793 (2004)
2. G.H. Hardy, E.M. Wright, *An Introduction to the Theory of Numbers*, 4. Aufl. (Oxford, 1960), S. 223–225
3. F.T. Leighton, *Introduction to Parallel Algorithms and Architectures* (Morgan Kaufmann, Burlington, 1992)
4. M.J. Quinn, *Parallel Computing: Theory and practice*, 2. Aufl. (McGraw-Hill, New York, 1994)

5. P.L. Montgomery, Modular multiplication without trial division. *Math. Comput.* **44**, 519–521 (1985)
6. J.-C. Bajard, L.-S. Dider, An RNS Montgomery modular multiplication algorithm, in *Proceedings. IEEE Symposium on Computer Arithmetic* (1997), S. 234–239
7. S.E. Eldridge, C.D. Walter, Hardware implementation of Montgomery’s modular multiplication algorithm. *IEEE Trans. Comput.* 693–699 (1993)
8. D.E. Knuth, *The Art of Computer Programming*. Seminumerical Algorithms, Bd. 2, 2. Aufl. (Addison-Wesley, Boston, 1981)
9. D.H. Lehmer, Computer technology applied to the theory of numbers, in *Studies in Number Theory*. MAA Studies in Mathematics (1969), S. 117–151
10. E.F. Brickell, A survey of hardware implementations of RS, in *Advances in Cryptology - CRYPTO’89. Proceedings, CRYPTO 1989*, ed. by G. Brassard. Lecture Notes in Computer Science, Bd. 435 (1990), S. 368–370
11. D.M. Chiarulli, W.G. Rudd, D.A. Buell, DRAFT-a dynamically reconfigurable processor for integer arithmetic, in *IEEE 7th Symposium on Computer Arithmetic (ARITH)* (1985), S. 309–317
12. D.M. Chiarulli, A fast multiplier for very large operands, Technical Report (University of Pittsburgh Technical Report CSTR 86-11, 1986)
13. D.M. Chiarulli, A horizontally reconfigurable architecture for extended precision arithmetic. PhD thesis, Baton Rouge, Louisiana, 1986
14. W.G. Rudd, D.M. Chiarulli, D.A. Buell, A high performance factoring machine, in *Proceedings of 11th Annual International Symposium on Computer Architecture* (1984), pp. 297–300
15. A.F. Tenca, C.K. Koc, A scalable architecture for Montgomery multiplication, ed. by C. K. Koc, C. Paar. Lecture Notes in Computer Science, Bd. 1717 (1999), S. 94–108
16. S.E. Eldridge, A faster modular multiplication algorithm. *Int. J. Comput. Math.* 63–68 (1991)
17. T. Granlund, GNU MP: the GNU multiple precision arithmetic library (Free Software Foundation, Inc., 2001)
18. V. MüCller, Fast multiplication on elliptic curves over small fields of characteristic two. *J. Cryptol.* **11**, 219–234 (1998)
19. A. Schönhage, V. Strassen, Schnelle multiplikation grosser zahlen. *Computing* 281–292 (1971)
20. C.D. Walter, S.E. Eldridge, A verification of Brickell’s fast modular multiplication algorithm. *Int. J. Comput. Math.* 153–169 (1990)



Moderne symmetrische Chiffren – DES und AES

9

Zusammenfassung

In einem symmetrischen Kryptosystem ist der Schlüssel, der zum Verschlüsseln einer Nachricht verwendet wird, derselbe wie der Schlüssel, der zum Entschlüsseln einer Nachricht verwendet wird. Obwohl dies eine Belastung für die ordnungsgemäße Schlüsselverwaltung und -sicherheit für die Benutzer eines solchen Kryptosystems darstellt, wurden zwei wichtige Kryptosysteme, der Digital Encryption Standard (DES) und der Advanced Encryption Standard (AES), vom National Institute of Standards and Technology veröffentlicht. DES war aufgrund der Art und Weise, wie es veröffentlicht wurde, umstritten und wurde weitgehend vom AES abgelöst, über den fast keine Kontroversen bestehen. AES wird derzeit weit verbreitet eingesetzt, zum Teil weil es der NIST-Standard ist und zum Teil weil sein Design es schnell und auf einer Vielzahl von Plattformen mit unterschiedlichen Rechenkapazitäten nutzbar macht. Dieses Kapitel behandelt die technischen Aspekte von AES. Code für AES und Testergebnisse erscheinen im Anhang B, so dass Code-Tests durchgeführt und der Verschlüsselungsprozess beobachtet werden können.

9.1 Geschichte

Anfang der 1970er Jahre erkannte das United States National Bureau of Standards (NBS)¹, dass ein kryptographisch sicherer Algorithmus für die elektronische Kommunikation benötigt wurde. Die Suche begann 1972. Ein Vorschlag wurde 1975 veröffentlicht und wurde aus mehreren Gründen kritisiert. Dennoch wurde der Data Encryption Standard

¹ Der Name wurde 1988 in National Institute for Standards and Technology (NIST) geändert.

(DES) war am 15. Januar 1977 als FIPS (Federal Information Processing Standard) Publication 46 [1] veröffentlicht. Der Standard wurde mehrmals bestätigt und erst lange nach seiner erwarteten sicheren Nutzungsdauer aufgehoben. Ein Hauptgrund für die Einführung eines kryptographischen Standards war die Ermöglichung von Finanzinstituten, ihr Geschäft über gesicherte Kommunikationskanäle abzuwickeln; selbst in den späten 1970er und frühen 1980er Jahren wurden viele Banken² über unverschlüsselte Telefonleitungen mit den regionalen Federal Reserve Banken abgeglichen.

9.1.1 Kritik und Kontroverse

Der Vorläufer von DES war ein kryptographischer Algorithmus von IBM namens LUCIFER. Im Gegensatz zu LUCIFER, der einen 112-Bit-Schlüssel hatte, ist der Schlüssel für DES nur 56 Bit. Angesichts der Tatsache, dass die United States National Security Agency (NSA) bekanntermaßen an der Entwicklung von DES beteiligt war,³ gab es viele Verschwörungstheorien, dass DES absichtlich als eine Version von LUCIFER geschaffen wurde, die so geschwächt war, dass die NSA sie knacken konnte.

Fast sofort wurde ein Plan zum Knacken von DES vorgelegt [2]. Der Diffie-Hellman-Plan sah parallele Berechnungen vor, möglicherweise unterstützt durch spezielle Hardware. Eine Version war für einen 20-Millionen-Dollar-Computer, der einen DES-Schlüssel in 12 Stunden knacken würde. Im Wesentlichen war der Angriff ein paralleler Brute-Force-Angriff; das Aufteilen der 2^{56} möglichen Schlüssel in Blöcke und dann das Testen aller Schlüssel in einem gegebenen Block ist ein perfektes Beispiel für eine peinlich parallele Berechnung.

Weitere Kritik war, dass ein Teil des Verschlüsselungssystems (bekannt als die „S-Boxen“) absichtlich so erstellt wurde, dass es eine Gruppenoperation auf den Eingabebits durchführt. Wie später im Kontext von diskreten Logarithmen diskutiert wird, wenn ein Teil der Verschlüsselung eine Gruppenoperation ist

$$g : \text{bits} \rightarrow \text{other_bits}$$

dann gibt es ein Inverses g^{-1} in der Gruppe zur Operation g , und dieses Inverse könnte eine Hintertür sein, die es denen, die über die Gruppenstruktur Bescheid wissen (vermutlich die NSA), ermöglicht, ohne großen Aufwand die scheinbar kryptographisch wichtige Aktion der S-Boxen rückgängig zu machen.

²Einschließlich der eigenen des Autors, anscheinend.

³Das NIST ist eine Abteilung des Handelsministeriums, die mit der Erstellung von Standards beauftragt ist, die dem US-Handel zugutekommen. Das NIST verfügt über Expertise in der Kryptographie, ist aber generell gesetzlich verpflichtet, in technischen Fragen (wie der Kryptographie), für die die NSA die offizielle Regierungsbehörde ist, mit der NSA zusammenzuarbeiten.

Obwohl es in der menschlichen Natur liegt, dass wir von Verschwörungstheorien fasziniert sind, ist es nicht klar, ob diese irgendeine Substanz haben. Don Coppersmith hat zum Beispiel [3] gezeigt, dass die *S*-Boxen keine Gruppenstruktur haben. Sein Beweis war im Wesentlichen eine rechnerische Demonstration, dass, wenn es eine Gruppenstruktur gäbe, dann müsste die Ordnung der Gruppe viel größer sein als die größtmögliche Gruppe, die auf die Eingabebits wirken könnte.

So viel zur Gruppenstruktur.

Vielmehr war es tatsächlich so, dass die Electronic Frontier Foundation (EFF) DES geknackt hat, mit Hilfe von verteiltem parallelem Rechnen, in 22 Stunden und 15 Minuten, im Jahr 1999. Diese Leistung war völlig konsistent mit dem ursprünglichen Vorschlag, wie man einen 56-Bit-Schlüssel dieser Art mit paralleler Berechnung knacken kann, da diese Berechnung im Laufe der Zeit an Fähigkeit gewonnen hat.

Was von denjenigen, die Verschwörungstheorien lieben, jedoch übersehen zu werden scheint, ist, dass DES zwar mehrmals bestätigt wurde, der ursprüngliche Plan jedoch war, dass es ein Algorithmus sein sollte, der für 15 Jahre kryptographische Sicherheit bietet.

Eine Veröffentlichung von 1977 plus 15 Jahre bringt uns zu 1992, sieben Jahre bevor die EFF DES auf offensichtliche Weise knackt. Ja, DES kann geknackt werden, und ja, DES kann genau auf die Weise geknackt werden, wie es gezeigt wurde, als es verbreitet wurde, knackbar zu sein. Aber es gab viele Menschen und Institutionen mit einem gewissen Eigeninteresse daran zu demonstrieren, dass DES von Anfang an fehlerhaft war, und es ist überhaupt nicht klar, dass sie ihren Punkt gemacht haben [4, 5].

9.2 Der Advanced Encryption Standard

Teilweise aufgrund von Sicherheitsbedenken verwendeten viele, die in den 1990er Jahren DES nutzten, „Triple DES“, indem sie dreimal statt nur einmal verschlüsselten. Es war NIST Mitte der 1990er Jahre klar, dass ein neuer Verschlüsselungsstandard notwendig war. Trotz der Tatsache, dass die Verschwörungstheorien über DES nicht gerechtfertigt zu sein schienen, ging NIST bei der Verschlüsselungsdesignarbeit, die es 1997 mit dem Advanced Encryption Standard (AES)that begann, einen völlig anderen Weg. Dies sollte ein Algorithmus sein, der für sensible, aber nicht für klassifizierte Informationen verwendet wird.

Dieses Mal wurde sehr wenig im Geheimen getan. Tatsächlich wurde die Designbestimmung im Januar 1997 als Wettbewerb eröffnet.

Alle Anwärter wurden aufgefordert, vorgeschlagene Designs einzureichen, und alle Anwärter durften die Vorschläge implementieren und angreifen.

Eine Reihe von NIST-Kriterien wurden im Voraus skizziert. Das Ziel war ein Kryptosystem, das mindestens so sicher war wie das moderat weit verbreitete Triple DES, aber ein Kryptosystem, das viel effizienter war als Triple DES.

Die Einschränkungen hinsichtlich Geschwindigkeit und Implementierung auf Geräten mit geringer Leistungsfähigkeit erforderten, dass ernsthafte Anwärter auf solchen

Geräten implementierbar sein sollten. Der Rijndael/AES-Algorithmus ist beispielsweise stark byteorientiert, was ihn in einer Hochsprache auf einem Standardprozessor sauber und effizient macht, aber auch relativ einfach auf dem Typ eines Prozessors mit minimaler Leistungsfähigkeit zu implementieren, wie er auf einer Smartcard gefunden werden könnte. Die NIST-Spezifikation war für eine Blocklänge von 128 Bits und für Schlüssellängen von 128, 192 und 256 Bits.

Schließlich wurden strenge Standards bezüglich geistiger Eigentumsfragen festgelegt. Kein Vorschlag würde beispielsweise angenommen, wenn seine Implementierung und Nutzung durch geistige Eigentumsbeschränkungen eingeschränkt wären; NIST beabsichtigte, dass AES ein offener Standard ohne Belastungen durch Patente oder andere behauptete proprietäre Inhalte sein sollte.

Einreichungen wurden in einer Reihe von öffentlichen Konferenzen [6–8] gemacht und bewertet, bevor die endgültige Ankündigung 2001 gemacht wurde [9].

Für die erste AES-Bewertungsrunde wurden 15 ursprüngliche Einreichungen akzeptiert. Diese wurden auf einer Konferenz in Ventura, Kalifornien, im August 1988 vorgestellt und sind in Abb. 9.1 dargestellt. Eine zweite Konferenz fand im März 1999 statt, parallel zum jährlichen Fast Software Encryption Workshop, auf dem Forscher Implementierungen, Analysen und Kritiken der 15 Einreichungen vorstellten. Im August 1999 wurde die Liste der fünfzehn Kandidaten auf fünf reduziert – MARS, RC6, Rijndael, Serpent und Twofish, und Forscher präsentierten Arbeiten zu diesen fünf auf dem Fast Software Encryption Workshop im April 2000.

Die ursprünglichen drei Kriterien Sicherheit, Kosten und Implementierungsmerkmale wurden während des Bewertungsprozesses etwas modifiziert. Obwohl die Sicherheit nach wie vor das Hauptanliegen war, führte die Analyse der vorgeschlagenen Algorithmen zu einer Verfeinerung der anderen beiden Kriterien. Die Kostenkriterien

Cryptosystem	Submitter(s)
CAST-256	Entrust (Canada)
Crypton	Future Systems (KR)
DEAL	Outerbridge and Knudsen (USA and Denmark)
DFC	ENS-CNRS (France)
E2	NTT (Japan)
Frog	TecApro (CR)
HPC	Schroeppe (USA)
LOKI97	Brown, et al. (Australia)
Magenta	Deutsche Telekom (Germany)
MARS	IBM (USA)
RC6	RSA (USA)
Rijndael	Daemen and Rijmen (Belgium)
SAFER+	Cylink (USA)
Serpent	Anderson, Biham, Knudsen (UK, Israel, Denmark)
Two sh	Counterpane (USA)

Abb. 9.1 Die ursprünglichen AES-Kandidaten

umfassten sowohl die Softwareeffizienz als auch die Kosten, sowohl im allgemeinen Siliziumbereich als auch im benötigten Speicher, einer Hardwareimplementierung. Implementierungsmerkmale beinhalteten die Details der Implementierung in Silizium, in Field Programmable Gate Arrays und auf Allzweckprozessoren mit hohem Grad an Befehlsebenenparallelität. Es wurden auch die Flexibilität des Algorithmus berücksichtigt, Parameter außerhalb der ursprünglichen AES-Anforderungen aufzunehmen (falls Angriffe auf den ursprünglichen Algorithmus entdeckt wurden).

Am Ende dieser dritten Konferenz unterstützten die Kryptographen überwältigend Rijndael, die Einreichung von Vincent Rijmen und Joan Daemen. Die Auswahl von Rijndael als AES wurde in einer Pressemitteilung am 2. Oktober 2000 bekannt gegeben und durch die Veröffentlichung von FIPS-197 am 26. November 2001 gefolgt [9]. Die Sicherheit aller Finalisten wurde als ausreichend beurteilt. Im Allgemeinen kann die Wahl von Rijndael auf die Einfachheit der von ihm benötigten Operationen und die Byteorientierung dieser Operationen zurückgeführt werden. Diese führten zu einer relativ hohen Ausführungseffizienz sowohl in der Software als auch in der Hardware, obwohl der umfangreiche Gebrauch von Speichertabellen zu einer relativ großen Siliziumfläche unter den Finalistenalgorithmen führt. Schließlich beinhaltete Rijndael wie vorgeschlagen die Variationen in Schlüssel- und Blockgröße über die ursprünglichen Spezifikationen für AES hinaus, die in einem flexiblen Algorithmus benötigt würden.

Obwohl einige Schwächen in AES gefunden wurden, waren keine davon grundlegend oder führten zu Angriffen, die die Verwendung von AES aufgeben würden.

9.3 Der AES-Algorithmus

AES ist ein *schlüsselwechselnder Blockchiffre*, bei dem Klartext in Blöcken von 128 Bits verschlüsselt wird. Die Schlüsselgrößen in AES können 128, 192 oder 256 Bits betragen. Es handelt sich um einen *iterierten Blockchiffre* weil ein fester Verschlüsselungsprozess, normalerweise als *Runde* bezeichnet, mehrmals auf den Block von Bits angewendet wird. Schließlich meinen wir mit *schlüsselwechselnd*, dass der Chiffreschlüssel mit dem *Zustand* (die laufende Version des Blocks von Eingabebits) abwechselnd mit der Anwendung der Rundentransformation XORed wird.

Das ursprüngliche Rijndael-Design ermöglicht jede Wahl von Blocklänge und Schlüsselgröße zwischen 128 und 256 in Vielfachen von 32 Bits. In diesem Sinne ist Rijndael eine Obermenge von AES; die beiden sind nicht identisch, aber der Unterschied liegt nur in diesen Konfigurationen, die ursprünglich in Rijndael eingeführt, aber nicht in AES verwendet wurden.

Die einzige andere Unterscheidung, die zu beachten ist, liegt in der Kennzeichnung der Transformationen von AES. Wir werden der Kennzeichnung des FIPS und des Buches der Erfinder [10] folgen und nicht der des ursprünglichen Einreichens von Rijndael zum AES-Wettbewerb. Zum Beispiel bezeichnete die ursprüngliche Einreichung eine ByteSub-Transformation, und das FIPS bezieht sich jetzt auf SubBytes.

9.3.1 Polynomiale Grundlagen: Das Galois-Feld $GF(2^8)$

AES macht ausgiebigen Gebrauch von dem Polynom

$$m(x) = x^8 + x^4 + x^3 + x + 1,$$

des achten Grades und dem endlichen Feld von $2^8 = 256$ Elementen, die durch dieses Polynom erzeugt werden.

Wir stellen nun fest, dass wir auf die Polynomnotation verzichten und die Polynome einfach als Bitfolgen betrachten können, wobei das 8-Bit-Byte, das die Bitfolge $b_7b_6b_5b_4b_3b_2b_1b_0$ ist, als Kurznotation für das Polynom des siebten Grades

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$$

verwendet wird, das ein Element des endlichen Feldes von 256 Elementen ist, das durch $m(x)$ definiert ist.

AES macht ausgiebigen Gebrauch von der Tatsache, dass Bitfolgen byte-weise genommen und auf diese Weise als Koeffizienten von Polynomen interpretiert werden können. Entscheidend für die Leistung von AES ist die Tatsache, dass die von Null verschiedenen Polynome als Potenzen eines einzigen Polynoms $x + 1$ erzeugt werden können. Dies ermöglicht es uns, die Potenzen des Generators zu verwenden, um eine Tabelle von Logarithmen zu erstellen und die Multiplikation im Galois-Feld durch Tabellenabfrage durchzuführen. Tatsächlich besteht ein großer Teil dessen, was man verstehen muss, um Software zu verstehen, die AES implementiert, darin zu verstehen, dass die byte-orientierte Tabellenabfrage tatsächlich die anspruchsvollere Mathematik implementiert.

9.3.1.1 Mehr Polynomarithmetik

Als Teil der Verschlüsselungs- und Entschlüsselungsprozesse verwendet AES Gruppen von vier Bytes, um Polynome $f(X)$ dritten Grades in einer unbestimmten X zu definieren; jedes Byte wird als Definition eines Elements von $GF(2^8)$ genommen, das einer der Koeffizienten von $f(X)$ ist. Der AES-Prozess führt dann Arithmetik an diesen Polynomen modulo dem Polynom $X^4 + 1$ durch. Wir schreiben diese Polynome als

$$a_3 \odot X^3 + a_2 \odot X^2 + a_1 \odot X + a_0$$

und schreiben die Koeffizienten a_i in hexadezimaler Notation, so dass ein spezifischer Koeffizient $a_3 = x^5 + x^3 + 1 \leftrightarrow 00101001$ als 29 geschrieben würde.

Obwohl es eine mathematische Grundlage für diese Operationen gibt, können wir dies aus rechnerischer Sicht fast als eine Positionsschreibweise für die Arithmetik betrachten. Da $X^4 \equiv 1 \pmod{X^4 + 1}$, ist die Multiplikation eines Polynoms $f(X)$ mit $b \odot X$ modulo $X^4 + 1$ wirklich eine koeffizientenweise Multiplikation durch b in $GF(2^8)$ und eine links zirkuläre Verschiebung der Koeffizienten:

$$(b \odot X) \cdot (a_3 \odot X^3 + a_2 \odot X^2 + a_1 \odot X + a_0) \equiv \\ (b * a_2) \odot X^3 + (b * a_1) \odot X^2 + (b * a_0) \odot X (b * a_3) \pmod{X^4 + 1}$$

wo die Multiplikation $*$ der Koeffizienten in $GF(2^8)$ stattfindet. Es ist teilweise, um diese rein formale Natur dieser Polynome zu erkennen, dass wir das \odot Symbol für die Multiplikation durch die Koeffizienten verwenden.

9.3.2 Byte-Organisation

Die Eingabe für AES ist der *Klartext*, eine Sequenz von Blöcken von jeweils 128 Bits der zu verschlüsselnden Nachricht, und der *Schlüssel*, ein Block von $K = 128, 192$ oder 256 Bits, wobei die Größe eine Option des Benutzers ist. Die Blöcke des Klartexts werden mit dem Schlüssel verschlüsselt, um einen *Geheimtext* aus Blöcken von jeweils 128 Bits zu erzeugen. AES ist ein *symmetrischer* Chiffre, insofern als der durch Klartext und Schlüssel erzeugte Geheimtext mit demselben Schlüssel wieder in Klartext umgewandelt wird.

Vereinfacht betrachtet, ist AES fast (aber nicht ganz) eine äußere Schleife von N_r Iterationen, jede als *Runde* bezeichnet, von Bit-Transformationen und einem inneren Satz von vier Transformationsstufen pro Runde. Das aktuelle Bitmuster als Eingabe zu oder Ausgabe von einer dieser Transformationen wird als *Zustand* bezeichnet.

Der AES-Klartextblock ist 128 Bits lang. AES ist stark byteorientiert; wenn wir den Strom der Bytes sowohl des Klartexts als auch des Schlüssels als in aufsteigender Reihenfolge nummeriert betrachten

$$p_0 p_1 \dots p_{15}$$

und

$$k_0 k_1 \dots k_{K/8},$$

dann werden die Bytes sowohl des Klartexts als auch des Schlüssels normalerweise als zweidimensionales Array in spaltenorientierter Reihenfolge betrachtet, dargestellt für den Klartext in Abb. 9.2; der Schlüssel kann ähnlich dargestellt werden.

Ein Schlüssel würde in einem ähnlichen Muster von vier Reihen und $K/32 = 4, 6$ oder 8 Spalten angeordnet sein, jeweils für die Schlüssellängen von $K = 128, 192$ oder 256 Bits.

Abb. 9.2 Byte-für-Byte-Ansicht der 128 Bits eines Klartextblocks

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

9.4 Die Struktur von AES

9.4.1 Die äußere Struktur der Runden

Das Rijndael-Referenzbuch [10] enthält Code für AES und Testdaten in seinen Anhängen D und E. Der Code in diesen Anhängen unterscheidet sich leicht von dem, was im Text des Buches gezeigt wird. Wir präsentieren hier und in einem Anhang in diesem Buch (unser Anhang B) eine modifizierte Version der C-Implementierung aus Anhang E von [10]. Im Großen und Ganzen sind die Unterschiede einfach eine Umbenennung von Funktionen; ursprüngliche Namen im Rijndael-Vorschlag wurden im AES-Standard geändert. Eine Änderung ist weniger geringfügig – die Anhang E (und unsere) Implementierung speichert den erweiterten Schlüssel als dreidimensionales Array, wobei das äußere Subskript das Subskript für die Rundenzahl ist, während der Text des Referenzbuchs ein zweidimensionales Array verwendet, wobei der Schlüssel für jede Runde aus einer Spalte des zweidimensionalen Arrays stammt.

Ein substantielleres, aber leicht zu behebendes Problem mit dem Code von Anhang E besteht darin, dass die Entschlüsselungsfunktion, wie sie präsentiert wird, die Bit-Transformation in der gleichen Reihenfolge wie die Verschlüsselung durchführt, anstatt in umgekehrter Reihenfolge. Als symmetrischer Chiffre ist die AES-Entschlüsselung der gleiche Prozess wie die Verschlüsselung, jedoch in umgekehrter Richtung und mit einigen der Funktionen, die durch ihre Inversen ersetzt werden. Glücklicherweise ist es einfach zu wissen, dass man die Entschlüsselungsfunktion richtig gemacht hat, weil sie die Verschlüsselung umkehrt und den Chiffretext wieder in Klartext umwandelt.

Die äußere Struktur der Verschlüsselung mit AES wird in Abb. 9.3 gezeigt. Für Schlüssellängen von $K = 128, 192$ und 256 Bits verwenden wir $N_r = 10, 12$ und 14 Rundentransformationen entsprechend.

Abb. 9.3 Äußere
Verschlüsselungsstruktur von
AES

```
Rijndael(State, CipherKey) {
    KeyExpansion(CipherKey, ExpandedKey);
    AddRoundKey(State, ExpandedKey[0]);
    for( i = 1; i < Nr; i++) {
        SubBytes(State);
        ShiftRows(State);
        MixColumns(State);
        AddRoundKey(State, ExpandedKey[i]);
    }

    SubBytes(State);
    ShiftRows(State);
    AddRoundKey(State, ExpandedKey[Nr]);
}
```


9.4.2 Allgemeine Code-Details

Wir bemerken, dass die hier gezeigten Codefragmente und die in unserem Anhang B vollständiger gezeigten mehrere globale Definitionen verwenden, die wir in Abb. 9.4 auflisten. Die Konstante MAXBC ist die maximale Anzahl von 32-Bit-Subblöcken in einem Klartextblock, die für das allgemeinere Rijndael auf 8 gesetzt wird, aber für AES als 4 genommen würde. In der Implementierung der Daemen und Rijmen Referenz [10], ermöglicht die Variable BC auf 4, 6 oder 8 gesetzt, alle Rijndael-Optionen zu testen.

Die Konstante MAXKC ist die maximale Anzahl von 32-Bit-Subblöcken in einem Schlüssel, ebenfalls auf 8 gesetzt; Die Variable KC kann 4, 6 oder 8 für Schlüssel der Längen 128, 192 oder 256 sein.

Die Konstante MAXROUNDS ist auf 14 gesetzt, und der Wert ROUNDS für verschiedene Schlüssellängen wird mit dem globalen Array numrounds festgelegt.

9.4.3 Schlüsselerweiterung

Der Eingabeschlüssel wird zunächst mit der Funktion KeyExpansion erweitert, um einen Schlüssel zu erzeugen, der $N_r + 1$ mal so groß ist wie seine ursprüngliche Größe. Der erweiterte Schlüssel wird dann in Blöcken von K Bits auf einmal genommen. Ein Block wird dem Zustand vor den Rundeniterationen hinzugefügt, $N_r - 1$ Blöcke werden hinzugefügt, jeweils am Ende jeder der Runden in der Schleife, und der letzte Block wird als letzte der Transformationen hinzugefügt.

Die Schlüsseladditionsschritte erfordern eine erhebliche Anzahl von Schlüsselbits. Diese Bits werden durch einen Erweiterungsprozess aus dem ursprünglichen Schlüssel gewonnen.

```
typedef unsigned char word8;
typedef unsigned int word32;

#define MAXBC 8
#define MAXKC 8
#define MAXROUNDS 14

bool testd2, testd3, testtext;
int BC, KC, ROUNDS;

static int numrounds[5][5] = {{10, 11, 12, 13, 14},
                               {11, 11, 12, 13, 14},
                               {12, 12, 12, 13, 14},
                               {13, 13, 13, 13, 14},
                               {14, 14, 14, 14, 14}};
```

Abb. 9.4 (Einige der) globalen Definitionen

Natürlich muss bei der Erweiterung eines Eingabeschlüssels Vorsicht walten, da die resultierenden Schlüsselbits nicht mehr inhärente Zufälligkeit enthalten werden, als im ursprünglichen Schlüssel vor der deterministischen Erweiterung vorhanden ist.

Mit zehn, zwölf oder vierzehn Runden in AES benötigt der Algorithmus $128 \cdot 11 = 1408$, $128 \cdot 13 = 1664$ oder $128 \cdot 15 = 1920$ Bits Schlüssel, um den Schritt `AddRoundKey` durchzuführen. Ein 128-Bit-Block des Schlüssels wird vor der Iteration der Runden verwendet, und dann werden zusätzliche 128-Bit-Blöcke für jede Iteration innerhalb der Runden verwendet. Die Schlüsselbits werden über die Funktion `KeyExpansion` ermittelt, die auf den Anfangsschlüsselwert angewendet wird.

Für eine Version von AES mit N_r Runden kann der erweiterte Schlüssel als dreidimensionales Array betrachtet werden, wobei der äußere Index die Rundenzahl ist und der innere Teil des Arrays vier Zeilen und 4, 6 oder 8 Spalten des Schlüssels für Schlüssel der Länge 128, 192 oder 256 Bits sind. Dies ist die Darstellung des Schlüssels im Code im Anhang E des Referenzbuchs [10]. Der Text der Referenz betrachtet den Schlüssel jedoch als zweidimensionales Array (unter Verwendung der letzten beiden Dimensionen des 3-d-Arrays), und wir diskutieren die einfachere Form, die Abb. 9.7 ist, anstatt der 3-d-Version von Abb. 9.5.

Für eine Version von AES mit N_r Runden betrachten wir den erweiterten Schlüssel als zweidimensionales Array mit vier Zeilen und $4 \cdot (N_r + 1)$ Spalten, die wir als $W[0..3][0..4 \cdot (N_r + 1)]$ indizieren. Wenn wir N_k auf 4, 6 oder 8 setzen, je nachdem, ob die Schlüssellänge 128, 192 oder 256 Bits beträgt, dann erhält der erste $4 \times N_k$ Block den ursprünglichen Schlüssel in spaltenweiser Reihenfolge wie in Abb. 9.16, und der Schlüssel wird dann durch die Anwendung der unten detaillierten rekursiven Funktion erweitert. Spalten von Bytes des Schlüssels werden rekursiv erzeugt:

1. Wenn der Spaltensubskript $j \geq N_k$ weder 0 modulo N_k noch 4 modulo N_k für $N_k = 8$ ist, dann haben wir

$$\begin{bmatrix} W[0][j] \\ W[1][j] \\ W[2][j] \\ W[3][j] \end{bmatrix} = \begin{bmatrix} W[0][j - N_k] \\ W[1][j - N_k] \\ W[2][j - N_k] \\ W[3][j - N_k] \end{bmatrix} \oplus \begin{bmatrix} W[0][j - 1] \\ W[1][j - 1] \\ W[2][j - 1] \\ W[3][j - 1] \end{bmatrix}$$

2. Wenn $N_k = 8$ (256-Bit-Schlüssel) und der Spaltensubskript j ist 4 modulo 8, dann XOR wir die $(j - N_k)$ -te Spalte nicht mit der $(j - 1)$ -sten Spalte, sondern mit den Bits, die durch die erste Anwendung von S auf diese Spalte erhalten wurden. Das heißt, wir haben die untenstehenden Bitoperationen. Dabei ist S die kombinierte $GF(2^8)$ und affine Transformation, die in `SubBytes` verwendet wird.

$$\begin{bmatrix} W[0][j] \\ W[1][j] \\ W[2][j] \\ W[3][j] \end{bmatrix} = \begin{bmatrix} W[0][j - N_k] \\ W[1][j - N_k] \\ W[2][j - N_k] \\ W[3][j - N_k] \end{bmatrix} \oplus \begin{bmatrix} S_{RD}(W[0][j - 1]) \\ S_{RD}(W[1][j - 1]) \\ S_{RD}(W[2][j - 1]) \\ S_{RD}(W[3][j - 1]) \end{bmatrix}$$

```

int KeyExpansion(word8 k[4][MAXKC], word8 W[MAXROUNDS+1][4][MAXBC]) {
    // Calculate the required round keys.
    int i, j, t, RCpointer = 1;
    word8 tk[4][MAXKC];
    for (j = 0; j < KC; j++) {
        for (i = 0; i < 4; i++) {
            tk[i][j] = k[i][j];
        }
    }
    t = 0;
    // Copy values into round key array.
    for (j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++) {
        for (i = 0; i < 4; i++) {
            W[t / BC][i][t % BC] = tk[i][j];
        }
    }
    while (t < (ROUNDS+1)*BC) {
        // While not enough round key material calculated, calc new values.
        for (i = 0; i < 4; i++) {
            tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
        }
        tk[0][0] ^= RC[RCpointer++];
        if (KC <= 6) {
            for (j = 1; j < KC; j++) {
                for (i = 0; i < 4; i++) {
                    tk[i][j] ^= tk[i][j-1];
                }
            }
        }
        else {
            for (j = 1; j < 4; j++) {
                for (i = 0; i < 4; i++) {
                    tk[i][j] ^= tk[i][j-1];
                }
            }
            for (i = 0; i < 4; i++) {
                tk[i][4] ^= S[tk[i][3]];
            }
            for (j = 5; j < KC; j++) {
                for (i = 0; i < 4; i++) {
                    tk[i][j] ^= tk[i][j-1];
                }
            }
        }
    }
    // Copy values into round key array.
    for (j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++) {
        for (i = 0; i < 4; i++) {
            W[t / BC][i][t % BC] = tk[i][j];
        }
    }
    return 0;
}

```

Abb. 9.5 Schlüsselerweiterung

3. Wenn der Spaltenunterindex i 0 modulo N_k ist, dann haben wir die unten aufgeführten Bitoperationen. Zusätzlich zur S_{RD} Operation haben wir eine zirkuläre Verschiebung nach unten der Bytes der Spalte $j - 1$ vor der Anwendung von S_{RD} und dem XOR in Byte 0 einer **Rundenkonstante** RC , wo

$$\begin{bmatrix} W[0][j] \\ W[1][j] \\ W[2][j] \\ W[3][j] \end{bmatrix} = \begin{bmatrix} W[0][j - N_k] \\ W[1][j - N_k] \\ W[2][j - N_k] \\ W[3][j - N_k] \end{bmatrix} \oplus \begin{bmatrix} S_{RD}(W[1][j - 1]) \oplus RC[j/N_k] \\ S_{RD}(W[2][j - 1]) \\ S_{RD}(W[3][j - 1]) \\ S_{RD}(W[0][j - 1]) \end{bmatrix}$$

$$RC[1] = x^0 \quad \text{that is, } 01$$

$$RC[2] = x^1 \quad \text{that is, } 02$$

...

$$RC[j] = x^{j-1} \quad \text{in } GF(2^8)$$

und wie immer können wir die $GF(2^8)$ Arithmetik mit Suchtabellen durchführen, in diesem Fall die Tabelle von Abb. 9.6:

Der `ExpandedKey[i]` Wert, wie er in der Pseudocode-Beschreibung des Algorithmus verwendet wird, bezieht sich auf die Spalten $N_b \cdot i$ bis $N_b \cdot (i + 1) - 1$ wenn sie als Spalten betrachtet werden, oder Bytes $4 \cdot N_b \cdot i$ bis $4 \cdot N_b \cdot (i + 1) - 1$ in spaltenweiser Reihenfolge genommen. Daher werden Schlüsselbits aus dem `ExpandedKey` in Blöcken von 128 Bits auf einmal extrahiert, aber die Schlüsselbits werden spaltenweise erzeugt, wie benötigt, nicht unbedingt in Blöcken von 128 Bits.

Insbesondere für Schlüssellängen von 128 oder 192 Bits wird der `ExpandedKey` mit der Funktion von Abb. 9.7 erstellt. Für Schlüssellängen von 256 Bits wird der `ExpandedKey` mit der Funktion von Abb. 9.8 erstellt.

9.4.4 SubBytes

9.4.4.1 Verschlüsselung

Der `SubBytes`-Schritt ist nichtlinear und ist tatsächlich der einzige nichtlineare Schritt in AES. Jedes einzelne Byte

$$a = a_7a_6a_5a_4a_3a_2a_1a_0$$

$$\begin{aligned} \text{word32 } RC[30] = \{ & 0x00, 0x01, 0x02, 0x04, 0x08, 0x10, \\ & 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, \\ & 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, \\ & 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, \\ & 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5 \}; \end{aligned}$$

Abb. 9.6 Die RC Suchtabelle für $GF(2^8)$ Arithmetik in der Schlüsselerweiterung

```

def KeyExpansion(byte K[4][Nk], byte W[4][Nb*(Nr+1)]):
    for j in range(0, Nk):
        for i in range(0, 4):
            W[i][j] = K[i][j]

    for j in range(Nk, Nb*(Nr+1)):
        if j % Nk == 0:
            W[0][j] = W[0][j-Nk] XOR S[W[1][j-1]] XOR RC[j/Nk]
            for i in range(1, 4):
                W[i][j] = W[i][j-Nk] XOR S[W[i+1 % 4][j-1]]
        else:
            for i in range(0, 4):
                W[i][j] = W[i][j-Nk] XOR W[i][j-1]

```

Abb. 9.7 Schlüsselerweiterung für 128- oder 192-Bit-Schlüssel (2-dimensionale Version)

(als Bitfolge geschrieben) des Zustands wird (zumindest konzeptionell) einer zweistufigen Transformation

$$a \rightarrow a^{-1} \text{ in } GF(2^8) \rightarrow f(a^{-1})$$

unterzogen, bei der $y = f(x)$ die Transformation von Abb. 9.9 und die $GF(2^8)$ Arithmetik ist durch das Polynom $m(x)$ oben definiert.

9.4.4.2 Entschlüsselung

Für die Entschlüsselung würde das Inverse zu SubBytes mit der Funktion f^{-1} von Abb. 9.10 gefolgt von einer Byte-Inversion in $GF(2^8)$ erreicht. Tatsächlich kann es mit dem gleichen C-Code, aber mit einer Umkehrtabelle, die invers zur Tabelle bei der Verschlüsselung ist, durchgeführt werden.

9.4.4.3 Implementierung – Die S-Funktion

Ein entscheidendes Merkmal von AES ist, dass seine Vorliebe für Berechnungen auf Bytes eine effiziente Implementierung ermöglicht. Obwohl die SubBytes-Operation konzeptionell eine Galois-Feldinversion gefolgt von einer affinen Transformation ist, können diese beiden kombiniert und mit einer 256 langen Tabellenabfrage implementiert werden, die die Funktion S im Code in unserem Anhang B und in Abb. 9.11 dargestellt ist. Die inverse Operation, die bei der Entschlüsselung verwendet wird, ist die gleiche Funktion, aber mit der inversen Suchtabelle S_i .

Für Hochsprachencode oder für die Implementierung auf jedem Standardprozessor ist dies fast sicher der effizienteste Ansatz, da die Intra-Wort-Bitmanipulationen der Galois-Inversion und der affinen Transformation nicht von CPU-Anweisungen unterstützt werden. Für Hardware-Implementierungen ist die Implementierung der tatsächlichen Arithmetik nicht ausgeschlossen, wie wir später diskutieren werden.

```

def KeyExpansion(byte K[4][Nk], byte W[4][Nb*(Nr+1)]):
    for j in range(0, Nk):
        for i in range(0, 4):
            W[i][j] = K[i][j]
    for j in range(Nk, Nb*(Nr+1)):
        if j % Nk == 0:
            W[0][j] = W[0][j-Nk] XOR S[W[1][j-1]] XOR RC[j/Nk]
            for i in range(1, 4):
                W[i][j] = W[i][j-Nk] XOR S[W[i+1 % 4][j-1]]
        else:
            for i in range(0, 4):
                W[i][j] = W[i][j-Nk] XOR W[i][j-1]

KeyExpansion(byte K[4][Nk], byte W[4][Nb*(Nr+1)])
{
    for(j = Nk; j < Nb*(Nr+1); j++) } /* expansion loop on columns */
{
    if(0 == j mod Nk) /* if-then for bytes down columns */
    {
        W[0][j] = W[0][j-Nk] XOR S[W[1][j-1]] XOR RC[j/Nk];
        for(i = 1; i < 4 i++)
        {
            W[i][j] = W[i][j-Nk] XOR S[W[i+1 mod 4][j-1]];
        }
    }
    else if(4 == j mod Nk)
    {
        for(i = 0; i < 4 i++)
        {
            W[i][j] = W[i][j-Nk] XOR S[W[i][j-1]];
        }
    }
    else
    {
        for(i = 0; i < 4 i++)
        {
            W[i][j] = W[i][j-Nk] XOR W[i][j-1];
        }
    } /* end if-then down columns */
} /* end expansion loop on columns */
} /* end KeyExpansion */

```

Abb. 9.8 Schlüsselerweiterung für 256-Bit-Schlüssel (2-dimensionale Version)

Abb. 9.9 Die Funktion $f(x)$ in SubBytes

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix}$$

Abb. 9.10 Die Funktion $f^{-1}(x)$ zum Invertieren von SubBytes

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix}$$

```
void SubBytes(word8 a[4][MAXBC], word8 box[255]) {
    // Replace every byte of the input by the byte
    // at that place in the non-linear S-box.
    int i, j;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] = box[a[i][j]];
        }
    }
}
```

Abb. 9.11 SubBytes

9.4.5 ShiftRows

9.4.5.1 Verschlüsselung

Die zweite Stufe der inneren Schleife von AES ist die Operation ShiftRows. In dieser Stufe werden die Bytes der vier Zeilen des Zustands zirkulär nach links verschoben. Zeile 0 des Zustands wird nicht verschoben; Zeile 1 wird um ein Byte nach links verschoben, Zeile 2 um zwei Bytes und Zeile 3 zirkulär um drei Bytes nach links. Ein grafisches Tableau für ShiftRows ist in Abb. 9.12 dargestellt. Der Code für die Funktion ist in Abb. 9.13 gezeigt.

Abb. 9.12 Byte-Transformationen des Schritts ShiftRows



```
// (Global value included here for completeness.)
static word8 shifts[5][4] = {{0, 1, 2, 3},
                             {0, 1, 2, 3},
                             {0, 1, 2, 3},
                             {0, 1, 2, 4},
                             {0, 1, 3, 4}};

void ShiftRows(word8 a[4][MAXBC], word8 d) {
    // Row 0 remains unchanged.
    // The other three rows are shifted a variable amount.
    word8 tmp[MAXBC];
    int i, j;

    if (d == 0) {
        for (i = 1; i < 4; i++) {
            for (j = 0; j < BC; j++) {
                tmp[j] = a[i][(j + shifts[BC-4][i]) % BC];
            }
            for (j = 0; j < BC; j++) {
                a[i][j] = tmp[j];
            }
        }
    }
    else {
        for (i = 1; i < 4; i++) {
            for (j = 0; j < BC; j++) {
                tmp[j] = a[i][(BC + j - shifts[BC-4][i]) % BC];
            }
            for (j = 0; j < BC; j++) {
                a[i][j] = tmp[j];
            }
        }
    }
}
```

Abb. 9.13 ShiftRows

9.4.5.2 Entschlüsselung

Bei der Entschlüsselung ist das Inverse des Schritts ShiftRows einfach die entsprechende zirkuläre Rechtsverschiebung der Bytes des Zustands.

9.4.6 MixColumns

9.4.6.1 Verschlüsselung

In der Stufe `SubBytes` wurden die Bits

$$b_7b_6b_5b_4b_3b_2b_1b_0$$

eines Bytes als Koeffizienten eines Polynoms

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

betrachtet, das ein Element des endlichen Feldes $GF(2^8)$ darstellte, und dieses Element wurde in $GF(2^8)$ invertiert. In der Stufe `MixColumns` führen wir diese Darstellung einen Schritt weiter. Die vier Bytes einer Spalte im Zustand werden jeweils als Elemente von $GF(2^8)$ betrachtet, die nun Koeffizienten eines kubischen Polynoms sind. Zum Beispiel würde eine Spalte des Zustands (mit Bytes als zwei hexadezimale Ziffern geschrieben)

1F
3D
5B
79

als Darstellung des Polynoms

$$1F \odot X^3 + 3D \odot X^2 + 5B \odot X + 79$$

genommen, wobei zum Beispiel der letzte Koeffizient $79 = 01111001$ als Darstellung des Polynoms

$$0 \cdot x^7 + 1 \cdot x^6 + 1 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$$

als Element von $GF(2^8)$ verstanden wird.

Die Spalten des Zustands, betrachtet als Polynome in X mit Koeffizienten in $GF(2^8)$, werden multipliziert mit

$$c(X) = 03 \odot X^3 + 01 \odot X^2 + 01 \odot X + 02$$

modulo $X^4 + 1$. Das Polynom $c(X)$ ist invertierbar modulo $X^4 + 1$, mit dem Inversen

$$d(X) = c^{-1}(X) = 0B \odot X^3 + 0D \odot X^2 + 09 \odot X + 0E.$$

Da dies komplizierter ist als die meisten Stufen von AES, werden wir etwas detaillierter darauf eingehen. Nehmen wir an, wir haben eine Spalte des Zustands

a_3
a_2
a_1
a_0

Dann ist die Multiplikation von MixColumns

$$(a_3 \odot X^3 + a_2 \odot X^2 + a_1 \odot X + a_0) \cdot (03 \odot X^3 + 01 \odot X^2 + 01 \odot X + 02)$$

welche umgeschrieben wird als

$$\begin{aligned} X^6. & (& & & & 03 \odot a_3) \\ X^5. & (& & & & 03 \odot a_2 + 01 \odot a_3) \\ X^4. & (& & & +03 \odot a_1 + 01 \odot a_2 + 01 \odot a_3) \\ X^3. & (03 \odot a_0 + 01 \odot a_1 + 01 \odot a_2 + 02 \odot a_3) \\ X^2. & (01 \odot a_0 + 01 \odot a_1 + 02 \odot a_2 & &) \\ X^1. & (01 \odot a_0 + 02 \odot a_1 & &) \\ X^0. & (02 \odot a_0 & &) \end{aligned}$$

and which reduces to

$$\begin{aligned} X^5. & (& & & & 03 \odot a_2 + 01 \odot a_3) \\ X^4. & (& & & +03 \odot a_1 + 01 \odot a_2 + 01 \odot a_3) \\ X^3. & (03 \odot a_0 + 01 \odot a_1 + 01 \odot a_2 + 02 \odot a_3) \\ \equiv X^2. & (01 \odot a_0 + 01 \odot a_1 + 02 \odot a_2 + 03 \odot a_3) \\ X^1. & (01 \odot a_0 + 02 \odot a_1 & &) \\ X^0. & (02 \odot a_0 & &) \end{aligned}$$

then to

$$\begin{aligned} X^4. & (& & & +03 \odot a_1 + 01 \odot a_2 + 01 \odot a_3) \\ X^3. & (03 \odot a_0 + 01 \odot a_1 + 01 \odot a_2 + 02 \odot a_3) \\ \equiv X^2. & (01 \odot a_0 + 01 \odot a_1 + 02 \odot a_2 + 03 \odot a_3) \\ X^1. & (01 \odot a_0 + 02 \odot a_1 + 03 \odot a_2 + 01 \odot a_3) \\ X^0. & (02 \odot a_0 & &) \end{aligned}$$

and finally to

$$\begin{aligned} X^3. & (03 \odot a_0 + 01 \odot a_1 + 01 \odot a_2 + 02 \odot a_3) \\ X^2. & (01 \odot a_0 + 01 \odot a_1 + 02 \odot a_2 + 03 \odot a_3) \\ \equiv X^1. & (01 \odot a_0 + 02 \odot a_1 + 03 \odot a_2 + 01 \odot a_3) \\ X^0. & (02 \odot a_0 + 03 \odot a_1 + 01 \odot a_2 + 01 \odot a_3) \end{aligned}$$

wo die Reduktion modulo $X^4 + 1$ erfolgt.

Die gesamte Operation an den Spalten des Zustands kann somit als Matrixmultiplikation in $GF(2^8)$ durchgeführt werden:

$$\begin{pmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 03 & 01 & 01 & 02 \\ 01 & 01 & 02 & 03 \\ 01 & 02 & 03 & 01 \\ 02 & 03 & 01 & 01 \end{pmatrix} \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

9.4.6.2 Entschlüsselung

Das Inverse zu MixColumns, genannt InvMixColumns, ist eine Multiplikation der Spalten mit dem Inversen $d(X)$, alles modulo $X^4 + 1$. Wie oben kann die Operation in eine Matrixoperation an den Spalten des Zustands wie folgt kondensiert werden.

$$\begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix}$$

9.4.6.3 Implementierung

Der Code für MixColumns und für InvMixColumns wird in den Abb. 9.14 und 9.15 dargestellt.

```
word8 mul(word8 a, word8 b) {
    // multiply two elements of GF(256)
    // required for MixColumns and InvMixColumns
    if (a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
    else return 0;
}

void MixColumns(word8 a[4][MAXBC]) {
    // Mix the four bytes of every column in a linear way.
    word8 b[4][MAXBC];
    int i, j;

    for (j = 0; j < BC; j++) {
        for (i = 0; i < 4; i++) {
            b[i][j] = mul(2, a[i][j])
                ^ mul(3, a[(i+1)%4][j])
                ^ a[(i+2)%4][j]
                ^ a[(i+3)%4][j];
        }
    }
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] = b[i][j];
        }
    }
}
```

Abb. 9.14 Code für die Funktion MixColumns

```

void InvMixColumns(word8 a[4][MAXBC]) {
    // Mix the four bytes of every column in a linear way.
    // This is the opposite operation of MixColumns.
    word8 b[4][MAXBC];
    int i, j;

    for (j = 0; j < BC; j++) {
        for (i = 0; i < 4; i++) {
            b[i][j] = mul(0xe, a[i][j])
                ^ mul(0xb, a[(i+1)%4][j])
                ^ mul(0xd, a[(i+2)%4][j])
                ^ mul(0x9, a[(i+3)%4][j]);
        }
    }
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] = b[i][j];
        }
    }
}

```

Abb. 9.15 Code für die Funktion InvMixColumns

$$\begin{array}{|c|c|c|c|} \hline p_0 & p_4 & p_8 & p_{12} \\ \hline p_1 & p_5 & p_9 & p_{13} \\ \hline p_2 & p_6 & p_{10} & p_{14} \\ \hline p_3 & p_7 & p_{11} & p_{15} \\ \hline \end{array}
 \oplus
 \begin{array}{|c|c|c|c|} \hline k_0 & k_4 & k_8 & k_{12} \\ \hline k_1 & k_5 & k_9 & k_{13} \\ \hline k_2 & k_6 & k_{10} & k_{14} \\ \hline k_3 & k_7 & k_{11} & k_{15} \\ \hline \end{array}
 =
 \begin{array}{|c|c|c|c|} \hline p'_0 & p'_4 & p'_8 & p'_{12} \\ \hline p'_1 & p'_5 & p'_9 & p'_{13} \\ \hline p'_2 & p'_6 & p'_{10} & p'_{14} \\ \hline p'_3 & p'_7 & p'_{11} & p'_{15} \\ \hline \end{array}$$

Abb. 9.16 AddRoundKey arbeitet auf den 128 Bits eines Klartextblocks

9.4.7 AddRoundKey

Der Schlüsseladditionsschritt wird als AddRoundKey bezeichnet. Da dies ein XOR der Bits des erweiterten Schlüssels mit dem Zustand ist, ist der Schritt AddRoundKey seine eigene Inverse. Die Schlüsseladdition wird in Abb. 9.16 dargestellt; der Code selbst wird in Abb. 9.17 angezeigt.

9.5 Implementierungsprobleme

AES wurde so konzipiert, dass es auf einer Reihe von Prozessoren gut funktioniert, einschließlich Smartcards mit kleinen 8-Bit-Prozessoren, schnellen Standardprozessoren und sogar auf spezieller Hardware. Da die Funktionen von AES Bitmanipulationen sind und viele dieser Funktionen nicht in der Instruction Set Architecture (ISA) eines Standardprozessors bereitgestellt werden, muss in einer Implementierung auf einem Standardprozessor eine

```

void AddRoundKey(word8 a[4][MAXBC], word8 rk[4][MAXBC]) {
    // XOR corresponding text input and round key input bytes.
    int i, j;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] ^= rk[i][j];
        }
    }
}

```

Abb. 9.17 Code für die Funktion AddRoundKey

Anpassung für die Bitverarbeitung vorgenommen werden. Andererseits wurde AES so konzipiert, dass diese Anpassungen relativ unkompliziert sind und dass selbst auf relativ leistungsschwachen Prozessoren eine hohe Leistung erzielt werden kann.

Um die notwendigen Operationen zu überprüfen, fassen wir die auszuführenden Operationen zusammen:

1. **KeyExpansion**: Die meisten der Schlüsselerweiterungsoperationen sind XORs. Die andere Operation ist die Anwendung der *S*-Funktion von SubBytes.
2. **SubBytes**: Mathematisch beinhaltet die Berechnung in SubBytes die $GF(2^8)$ Arithmetik gefolgt von der affinen Transformation $f(x)$. Rechnerisch kann dies alles durch Tabellensuche in einer 256 langen Tabelle durchgeführt werden und wird als *S*-Funktion bezeichnet.
3. **ShiftRows**: Dies besteht ausschließlich aus byte-orientierten Speicherverschiebungen des Zustandsarrays.
4. **MixColumns**: Mathematisch beinhaltet die MixColumns-Operation modulare Polynomoperationen mit Polynomen in X , deren Koeffizienten Elemente von $GF(2^8)$ sind. Rechnerisch ist die Polynomarithmetik nur Speicherverschiebungen nach der Arithmetik auf den Koeffizienten in $GF(2^8)$. Im Fall der Verschlüsselung ist die Koeffizientenarithmetik sehr einfach, da man nur Koeffizienten mit 1, x und $x + 1$ multiplizieren muss. Im Fall der Entschlüsselung sind die Multiplikatoren komplizierter und die Arithmetik ist daher schwieriger in Hardware zu implementieren. Im Fall einer Softwareimplementierung ist keine der Operationen kompliziert, da die Multiplikation normalerweise mit einer Tabellensuche durchgeführt wird.
5. **AddRoundKey**: Diese Operation ist einfach ein XOR des Schlüssels für die Runde und des Zustands.

9.5.1 Softwareimplementierungen

Die Hauptpunkte der Sorge für jede Softwareimplementierung kommen eindeutig auf drei Berechnungen herunter.

1. Die $GF(2^8)$ Arithmetik, die an mehreren Stellen erscheint.
2. Die byte-orientierten endlichen Feldoperationen in MixColumns.
3. Die Frage der Speicherung und/oder des Zugriffs auf die erweiterten Schlüsselbits.

Da AES ausschließlich auf Bytes operiert, können wir die XOR-Operationen und die Byte-Bewegungen des ShiftRows-Schritts ignorieren; es gibt hier keine Operationen, die nicht gut von der ISA eines Standardprozessors unterstützt werden.

Wir haben bereits darauf hingewiesen, dass die kombinierte SubBytes-Operation durch eine Tabellenabfrage mit der S-Funktion durchgeführt werden kann. Wenn dies nicht der Fall ist, muss man an anderen Stellen in der Berechnung in der Lage sein, Arithmetik in $GF(2^8)$ durchzuführen. Glücklicherweise kann dies mit festen arithmetischen Schritten erfolgen und benötigt keine komplexen Schleifen mit Entscheidungen. Das Polynom-Modul ist

$$m(x) = x^8 + x^4 + x^3 + x + 1,$$

so haben wir

$$\begin{aligned} x \cdot \sum_{i=0}^7 a_i x^i &= \sum_{i=0}^7 a_i x^{i+1} \\ &\equiv a_6 x^7 + a_5 x^6 + a_4 x^5 + (a_3 \oplus a_7) x^4 + (a_2 \oplus a_7) x^3 + a_1 x^2 + \\ &\quad (a_0 \oplus a_7) x^1 + a_7 \pmod{m(x)} \end{aligned}$$

Das Modul $m(x)$ ist ein 9-Bit-Muster $1|00011011$. Die Multiplikation des 8-Bit-Musters $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$ erzeugt das 9-Bit-Muster

$$a_7 | a_6 a_5 a_4 a_3 a_2 a_1 a_0 0,$$

so dass im Fall, dass $a_7 = 1$ wir die rechten 8 Bits mit einer Maske 00011011 XORieren, um die Reduktion modulo $m(x)$ durchzuführen. In der Software kann dies als ein Linksschieben implementiert werden, das möglicherweise von einem XOR mit einer Maske 00011011 gefolgt wird. Die Multiplikation mit jedem Element von $GF(2^8)$ kann erreicht werden, indem dieses Element in seine Potenzen von x zerlegt wird (effektiv durch die übliche rekursive Verdoppelung), so dass die grundlegende Operation der Multiplikation mit x (a.k.a. 02) als Kernel ausreicht.

Einer der Gründe für die Wahl des Polynoms $c(x)$ war, dass die Koeffizienten 01, 02 und 03 eine Multiplikation als einfache Operation ermöglichen. Die Multiplikation mit 01 ist in der Tat keine Multiplikation; die Multiplikation mit 02 ist die oben definierte Operation, und die Multiplikation mit 03 ist die Multiplikation mit 02 gefolgt von einem XOR. Leider sind die Koeffizienten 09, 0B, und 0D, und 0E des InvMixColumns-Schritts nicht von Natur aus so einfach, wenn nur, weil die nichttrivialen Einträge dichter sind und die Anzahl der 1-Bits größer ist, was mehr Bitoperationen für die $GF(2^8)$ -Operation erfordert.

Glücklicherweise hat, wie Daemen und Rijmen [10] hervorheben, P. Barreto beobachtet, dass die `InvMixColumns`-Multiplikation in zwei Matrixprodukte aufgeteilt werden kann, wie folgt.

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 04 & 04 & 00 & 05 \end{pmatrix}$$

Dies ermöglicht es, dass die `InvMixColumns` mit dem folgenden Vorverarbeitungsschritt implementiert werden kann, gefolgt von der gleichen Multiplikation, die in `MixColumns` verwendet wird.

Auf 32-Bit oder größeren Prozessorplattformen können die gleichen intrawort Operationen wie auf 8-Bit-Plattformen implementiert werden, aber die längere Wortlänge kann ein Vorteil sein, da man vier-Byte-Spalten in einem einzigen Schritt behandeln kann.

Software für AES ist relativ einfach zu implementieren, und die Verwendung der oben genannten Softwarefunktionen mildert wesentlich alle Komplexitäten aufgrund von ISA-Mängeln. Als Teil des ursprünglichen AES-Wettbewerbs und Auswahlprozesses war es notwendig, Referenzcode für jeden Algorithmus einzureichen. Referenzcode von P. Barreto und V. Rijmen erscheint in Daemen und Rijmen [10] und umfasst weniger als 350 Zeilen C, einschließlich vier Haupttabellen zur Suche der $GF(2^8)$ Arithmetik.

Mehrere Autoren haben während und kurz nach der Auswahl von AES die Verarbeitungsrate von Softwareimplementierungen von AES berichtet. Zeitmessungen sind bekanntermaßen schnell veraltet, und Zeitmessungen sind oft schwer zu vergleichen. Lipmaa berichtete [11] 260 Zyklen pro Verschlüsselung, oder 1,437 Gigabit pro Sekunde, für Verschlüsselung und 257 Zyklen pro Entschlüsselung, oder 1,453 Gigabit pro Sekunde, für Entschlüsselung, mit Assemblersprachenprogrammen auf einem 3,05 MHz Pentium 4 Prozessor, und 319 Zyklen (0,861 Gbit pro Sekunde) und 344 Zyklen (0,798 Gbit pro Sekunde) für Verschlüsselung und Entschlüsselung, jeweils mit C-Programmen (gcc 3.0.2) auf einem 2,25 MHz Athlon Prozessor. Andere Implementierungen werden zwischen 226 und 376 Zyklen auf geringeren Prozessoren berichtet, wobei die schnelleren Implementierungen in Assemblersprache und die langsameren Implementierungen in C oder C++ sind. Gladman berichtete ähnliche Zeitmessungen [12].

Es ist erwähnenswert, dass die Geschwindigkeit von AES in Software etwas langsamer ist als entweder DES oder Triple DES, aber nicht signifikant langsamer [13].

9.5.2 Hardware-Implementierungen

AES wurde so konzipiert, dass es für Smartcard- und ähnliche Anwendungen geeignet sein könnte. Daher sind, obwohl Softwareimplementierungen von Interesse sind,

die verschiedenen Hardware- oder programmierbaren Logikimplementierungen von AES von Interesse und neben der Geschwindigkeit werden Fragen der Siliziumressourcen und des damit verbundenen Stromverbrauchs relevant. Viele der Hardwareimplementierungen wurden vor der Annahme von Rijndael als AES durchgeführt und die Arbeiten wurden in den AES-Konferenzberichten veröffentlicht. Eine Reihe dieser Arbeiten bietet eine vergleichende Analyse der fünf Finalisten-Algorithmen. Einige Vergleiche wurden auch in anderen Zeitschriften oder Konferenzen veröffentlicht [14].

Die Hardware-Implementierungen von AES waren sehr unterschiedlich, teilweise aufgrund der vielen verschiedenen Anwendungsmöglichkeiten von AES. Viele dieser Implementierungen waren entweder spezifische ASICs oder ASIC-Designs; einige waren Architekturspezifikationen für einen Prozessor, der AES-Berechnungen im „nativ“ Modus unterstützen würde. Eine große Anzahl von Implementierungen wurde auf Field Programmable Gate Arrays (FPGAs) durchgeführt. Es wird weiterhin an algorithmischen Mitteln gearbeitet, mit denen die Verarbeitung beschleunigt werden könnte, unter der Annahme, dass man in der Hardware erhebliche Flexibilität bei der Manipulation der Bits hat; zu diesen Studien gehören einige über die beste Art und Weise, wie die Galois-Feldarithmetik in der Hardware unterstützt werden kann.

Obwohl die Hardware-Implementierungen unterschiedlich sind, kann man im Allgemeinen sagen, dass sie eine oder mehrere der folgenden Fragen beantworten.

1. Wenn man einen ASIC für AES entwerfen würde, welches Design würde den absolut schnellsten Durchsatz liefern?
2. Wenn man einen ASIC für AES entwerfen würde, welches Design würde den schnellsten Durchsatz liefern und nicht mehr Hardware verwenden, als auf einer Smartcard verfügbar sein könnte?
3. Wenn man einen ASIC für AES entwerfen würde, welches Design würde den schnellsten Durchsatz liefern und nicht mehr Hardware verwenden, als auf einer Netzwerkkarte verfügbar sein könnte?
4. Wenn man AES auf rekonfigurierbarer Hardware (FPGAs) implementieren würde, welches Design würde den absolut schnellsten Durchsatz liefern?
5. Wenn man AES auf rekonfigurierbarer Hardware implementieren würde, welches Design würde den schnellsten Durchsatz liefern und nicht mehr Hardware verwenden, als auf einer Smartcard verfügbar sein könnte?
6. Wenn man AES auf rekonfigurierbarer Hardware implementieren würde, welches Design würde den schnellsten Durchsatz liefern und nicht mehr Hardware verwenden, als auf einer Netzwerkkarte verfügbar sein könnte?

Die FPGA-basierten Implementierungen fügen der Definition von „best“ eine weitere Dimension hinzu, da sie es ermöglichen, eine Implementierung mit dem Aussehen und Gefühl eines ASIC zu entwerfen, die jedoch auf spezifischen kommerziellen Chips platziert werden muss. Wo Softwareimplementierungen durch die ISA des Prozessors eingeschränkt sind, sind die FPGA-Implementierungen durch die Größe und Art der

FPGA-Ressourcen eingeschränkt. In den meisten Fällen ist die letztendliche Einschränkung des Durchsatzes nicht auf den AES-Kern, sondern auf die Bandbreite durch das Gerät, von dem das FPGA ein Teil ist.

Darüber hinaus gibt es auf ASICs oder FPGAs Methoden zur Verbesserung der Leistung oder zur Verringerung der Größe durch Umordnung der Schritte des Algorithmus. Wenn die Hardwaregröße kein Problem ist, dann kann die iterative Schleife der Runden aufgerollt werden, um die Runden selbst zu pipelinen. Dies sollte eine erhöhte Durchsatzrate ermöglichen, auf Kosten einer Latenz, die im stationären Zustand nicht bemerkt wird, aber Hardware für jede einzelne Runde erfordert, anstatt ein einzelnes Hardwaremodul, das wiederholt verwendet wird.

Eine Auswirkung des Aufrollens der Schleife ist, dass die Anzahl der Lookup-Tabellen dramatisch ansteigen könnte, da man die Tabellen physisch nahe an der Logik halten möchte, die die gespeicherten Werte verwendet. Um die Hardwarekosten der $GF(2^8)$ Lookup-Tabellen zu vermeiden, kann man die Arithmetik in der Hardware durchführen; ein Vergleich zeigte eine sehr dramatische Abnahme der Hardwarenutzung und eine Erhöhung der Geschwindigkeit, als diese Änderung an einem Design vorgenommen wurde. Ein zusätzlicher Vorteil ist, dass der Speicherzugriff grundsätzlich sequenziell sein wird, was gegen die Parallelität der Hardware arbeitet, und die On-Chip-Speicherressourcen von FPGAs sind nicht ausreichend, um alle Tabellen bereitzustellen, die in einem vollständig aufgerollten AES-Design benötigt werden.

Selbst wenn die äußere Schleife der Runden nicht vollständig aufgerollt werden kann, besteht auch die Möglichkeit, in der Hardware den Verarbeitungsfluss innerhalb der Runden zu kombinieren. Im Allgemeinen gilt: Je größer die zu synthetisierende Hardware-Schaltung, desto effizienter und leistungsfähiger wird die Schaltung sein (bis die Schaltung so groß ist, dass die Werkzeuge nicht mehr ordnungsgemäß funktionieren können). Größere Designs bieten mehr Möglichkeiten für Synthesewerkzeuge, Parallelität zu extrahieren. Außerdem erfordert das Aufteilen eines großen Designs in Module oft Signale, die von einem Modul zum anderen übertragen werden müssen, sowohl bei der Ausgabe als auch bei der Eingabe registriert werden; wenn mehrere Module zusammen synthetisiert werden, dann können solche Signale ohne die künstliche Modularisierung behandelt werden.

9.6 Sicherheit

Der Hauptgrund für die Existenz eines kryptographischen Algorithmus besteht darin, die **Vertraulichkeit** von Daten zu gewährleisten, d. h. die Offenlegung von Daten gegenüber unbefugten Parteien zu verhindern. In seiner einfachsten Anwendung würde ein Benutzer eine Datei verschlüsseln, damit sie „klar“ übertragen werden kann, ohne dass befürchtet werden muss, dass der Inhalt von jemandem gelesen werden kann, der den Schlüssel nicht besitzt. Bewusstes Benutzerhandeln zur Verschlüsselung der Daten kann die erforderliche Sicherheit bieten, obwohl in einem Unternehmensumfeld die

Datenübertragungssoftware so konfiguriert werden könnte, dass dies transparent ist. Auf jeden Fall müssen die Daten nur einmal pro Übertragung in dieser End-to-End-Methode verschlüsselt und entschlüsselt werden, und die Verwaltung der Schlüssel ist in allen Szenarien am einfachsten, da die Schlüssel nur an die Benutzer verteilt werden müssen.

Eine kompliziertere Situation würde bestehen, wenn das Ziel darin bestünde, die Datenlasten einzelner Pakete zu verschlüsseln, nachdem der Übertragungsprozess begonnen hat, und wenn der Prozess der Entschlüsselung und erneuten Verschlüsselung an jedem Link entlang des Pfades vom Sender zum Empfänger stattfinden würde. Da die Anzahl der Pakete und die Anzahl der Links normalerweise jeweils viel größer als die Anzahl der übertragenen Dateien wäre und da der Prozess nun für die beteiligten Benutzer völlig transparent sein müsste, erfordert diese Situation eine viel höhere Geschwindigkeit der Verschlüsselung und Entschlüsselung. Dies erfordert auch einen viel anderen Standard für die Integrität des Schlüsselverteilungsprozesses, da alle Verbindungen von Link zu Link mit Schlüsseln versorgt werden müssen.

Unabhängig von der Anwendung ist die grundlegende Frage, die in Bezug auf jeden kryptographischen Algorithmus zu beantworten ist, „Ist er sicher?“ Die ersten Versuche der Kryptoanalyse, die im Rahmen des AES-Bewertungsprozesses durchgeführt wurden, sind im NIST-Bericht detailliert beschrieben. Es gab nachfolgende Arbeiten, die AES angegriffen haben, und eine Zusammenfassung einiger davon, vielleicht am prominentesten von Courtois, der eine Website [15] betreibt (oder betrieben hat). Courtois ist offensichtlich skeptisch gegenüber AES. Als Antwort auf die NESSIE (New European Schemes for Signatures, Integrity and Encryption) Pressemitteilung (2003), die besagt, dass keine Schwäche in AES (oder in 16 anderen Algorithmen, die dem europäischen Wettbewerb vorgelegt wurden) gefunden wurde, argumentiert Courtois „Das ist einfach nicht wahr und eine solche Empfehlung könnte ernsthafte Konsequenzen haben.“ Viel positiver, oder zumindest weniger skeptisch, über den Status von AES ist Landau [16], der schreibt „Die Kryptographie-Gemeinschaft ist eine ziemlich streitbare Gruppe, aber sie war nahezu einstimmig in ihrem Lob für NISTs AES-Bemühungen und die Wahl von Rijndael als Advanced Encryption Standard. Das ist in der Tat ein hohes Lob.“

Trotz der Beschwerden von Courtois scheint die Zukunft von AES also gesichert. Die NIST-Website sagt in der Antwort auf eine häufig gestellte Frage, dass AES „das Potenzial hat, weit über zwanzig Jahre hinweg sicher zu bleiben.“ Es scheint daher wahrscheinlich, dass AES für viele Jahre weiterhin ein zugelassener Algorithmus für die Nutzung durch die US-Regierung sein wird.

9.7 Übungen

1. (Programmierübung.) Überprüfen Sie, dass $x + 1$ tatsächlich von der Ordnung 256 modulo des Polynoms

$$m(x) = x^8 + x^4 + x^3 + x^1 + 1.$$

- ist.
2. (Mögliche Programmierübung.) Überprüfen Sie, dass $c(x)$ und $d(x)$ Inverse modulo $X^4 + 1$ sind.
 3. (Programmierübung.) Laden Sie den Code aus Anhang B herunter, kompilieren Sie ihn und überprüfen Sie, ob Sie die gleichen Ergebnisse wie in Anhang B erzielen.
 4. (Programmierübung.) Laden Sie den Code aus Anhang B herunter, kompilieren Sie ihn und überprüfen Sie, ob Sie die gleichen Ergebnisse wie in Anhang B erzielen.
 5. (Programmierübung.) Laden Sie den Code aus Anhang B herunter, kompilieren Sie ihn und überprüfen Sie, ob Sie ihn korrekt haben, indem Sie die Nachricht „this is the secret message“ verschlüsseln und dann entschlüsseln. Beachten Sie, dass dies zwei Blöcke (mit Padding auf dem zweiten) Text erfordert, da dies mehr als 16 Bytes lang ist.

Literatur

1. NIST, Fips 46-3: data encryption standard (reaffirmed) (1999), <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>
2. W. Diffie, M.E. Hellman, Exhaustive cryptanalysis of the NBS DES. *IEEE Comput.* 74–84 (1977)
3. D. Coppersmith, A.M. Odlyzko, R. Schroepel, Discrete logarithms in $GF(p)$. *Algorithmica* 1–15 (1986)
4. S. Landau, Standing the test of time: The Data Encryption Standard. *Not. Am. Math. Soc.* **47**, 341–349 (2000)
5. NIST, Data Encryption Standard (1999), <https://nvlpubs.nist.gov/nistpubs/sp958-lide/250-253.pdf>
6. NIST, in *1st AES Candidate Conference* (1998), <http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/aes1conf.htm>
7. NIST, in *2nd AES Candidate Conference* (1998), <http://csrc.nist.gov/CryptoToolkit/aes/round1/conf2/aes2conf.htm>
8. NIST, in *3rd AES Candidate Conference* (2000), <http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/aes3conf.htm>
9. NIST, FIPS 197: announcing the advanced encryption standard (AES) (2000), <http://csrc.nist.gov/encryption/aes/index.html>
10. J. Daemen, V. Rijmen, *The design of rijndael*, 2. Aufl. (Springer, Berlin, 2020)
11. H. Lipmaa, AES candidates: a survey of implementations (2004), <http://www.tcs.hut.fi/~helger/aes>
12. B. Gladman, Implementation experience with AES candidate algorithms, in *Proceedings, 2nd AES Candidate Conference* (1998)
13. C. Sanchez-Avila, R. Sanchez-Reillo, The Rijndael block cipher (AES proposal): a comparison with DES, in *Proceedings, 35th IEEE Carnahan Conference on Security Technology* (2001), S. 229–234
14. A.A. Dandalis, V.K. Prasanna, J.D. Rolim, A comparative study of performance of AES final candidate, in *Proceedings, 2nd International Workshop, Cryptographic Hardware and Embedded Systems*, Hrsg. von Ç.K. Koç, C. Paar. *Lecture notes in computer science*, Bd. 1965 (2000), S. 125–140

15. N.T. Courtois, Is AES a secure cipher? (2004) <http://www.cryptosystem.net/aes>. Accessed 17 Jun 2020
16. S. Landau, Communications security for the twenty-first century: the advanced encryption standard. *Not. Am. Math. Soc.* **47**, 450–459 (2000)

Zusammenfassung

Der Begriff eines asymmetrischen Verschlüsselungssystems stammt aus den 1970er Jahren, wobei die erste und immer noch primäre Version der asymmetrischen Verschlüsselung der RSA-Algorithmus von Rivest, Shamir und Adleman ist. Bei der asymmetrischen Verschlüsselung wird ein öffentlicher Schlüssel verwendet, um eine Nachricht zu verschlüsseln, die an den Besitzer des öffentlichen Schlüssels gesendet wird. Dieser Besitzer verwendet dann einen privat gehaltenen Schlüssel zum Entschlüsseln. Der RSA-Algorithmus basiert auf der Wahl von zwei großen Primzahlen p und q , die multipliziert werden, um einen Modulus $N=pq$ zu erzeugen. Der öffentliche Verschlüsselungsschlüssel e und der private Entschlüsselungsschlüssel d werden so gewählt, dass $ed \equiv 1 \pmod{\Phi(N)}$ gilt. Nach aktuellem Wissen der Mathematik ist es so, dass wenn N und e öffentlich sind, aber p , q und d privat gehalten werden, dann erfordert das Entschlüsseln einer Nachricht das Zerlegen von N in p mal q , und das ist rechnerisch schwierig. In diesem Kapitel legen wir die Grundlagen des RSA-Prozesses dar, mit einem Beispiel, und wir kommentieren die aktuellen Rekorde im Faktorisieren als Schätzung der Sicherheit von RSA.

10.1 Geschichte

AES ist derzeit der Stand der Technik in der symmetrischen Verschlüsselung. Der Algorithmus wurde in einem rigorosen Wettbewerb geprüft, von NIST als US-amerikanischer nationaler Standard veröffentlicht und hat sich als widerstandsfähig gegen alle Versuche, praktische Angriffe zu finden, erwiesen. Aber AES ist dennoch ein symmetrisches Verschlüsselungssystem, das sowohl vom Sender als auch vom Empfänger den Schlüssel kennt,

und dies erfordert daher, dass Sender und Empfänger über einen anderen, sicheren Mechanismus verfügen, um diesen Schlüssel zu übertragen.

Das Problem der sicheren Schlüsselverteilung hat im Laufe der Jahrhunderte alle geplagt, die Kryptographie verwenden würden. Jeder, der Kryptographie verwendet, muss immer daran denken, dass die beste verfügbare symmetrische Kryptographie völlig kompromittiert ist, wenn die Schlüssel nicht geheim gehalten wurden.

Es war also eine große neue Idee, als Whit Diffie und Martin Hellman ein Papier veröffentlichten, in dem sie argumentierten, dass asymmetrische Verschlüsselung möglich sein könnte, bei der der zur Verschlüsselung verwendete Schlüssel nicht derselbe ist wie der zur Entschlüsselung verwendete Schlüssel [1]. Diffie und Hellman schlugen ein Schema vor, bei dem ein öffentlich verfügbarer Schlüssel es jedem ermöglichen würde, eine Nachricht an einen Empfänger zu verschlüsseln, aber Verschlüsselung wäre nicht dasselbe wie Entschlüsselung, und es würde einen separaten, privaten Schlüssel erfordern, der nur vom Empfänger gehalten wird, um die Nachricht zu entschlüsseln.

Diffie und Hellman schlugen keinen Algorithmus vor, der diese „nicht-geheime“ Verschlüsselung ermöglichen könnte. Das kam fast sofort von Rivest, Shamir und Adleman, in ihrem bahnbrechenden Papier, das das jetzt als RSA-Algorithmus bekannte Verfahren beschreibt [2] und das fast (aufgrund der Bemühungen der US-Regierung) nie das Licht der Welt erblickte. Die beiden Papiere sind die klassischen öffentlichen Anfänge der Public-Key-Kryptographie.

Bis heute bleiben diese Autoren aufgrund ihrer ersten Veröffentlichungen diejenigen, die mit den Entdeckungen (oder Erfindungen?) in Verbindung gebracht werden. Es gibt jedoch eine gegenteilige Ansicht aus dem Vereinigten Königreich. Ende der 1990er Jahre veröffentlichte die britische Regierung Dokumente, die darauf hindeuten, dass ihre Forscher die ersten waren, die diese Ideen hatten. Die Arbeit von Diffie, Hellman, Rivest, Shamir und Adleman wurde in der Tat vielleicht bereits von Clifford Cocks, James Ellis und Malcolm Williamson, mit einiger Anregung von Nick Patterson, beim GCHQ (Government Communications Headquarters) in Großbritannien fast ein Jahrzehnt früher gemacht [3]. Die tatsächlichen Ursprünge der Public-Key-Kryptographie sind daher zur Diskussion gestellt, aber der Nutzen wird akzeptiert.

10.2 RSA Public-Key Verschlüsselung

Die grundlegende Frage für die Public-Key-Kryptographie ist diese: Gibt es einen Algorithmus, der die Veröffentlichung eines Verschlüsselungsschlüssels ermöglicht, den jeder sehen und verwenden kann, der eine „zufällige“ Bit-Version einer Nachricht erzeugt und der nur vom beabsichtigten Empfänger durch Verwendung eines vom Empfänger privat gehaltenen privaten Schlüssels entschlüsselt werden kann?

Die Antwort darauf von Rivest, Shamir und Adleman lautet „ja“, vorausgesetzt, wir akzeptieren die Annahmen, dass die Verschlüsselung schnell sein sollte, dass die Entschlüsselung durch diejenigen, die den privaten Schlüssel kennen, schnell sein sollte,

aber dass die Entschlüsselung (durch etwas möglicherweise Besseres als Brute-Force-Verschlüsselung) zumindest „rechnerisch unmöglich“ sein sollte, selbst für Gegner mit modernsten Rechenanlagen.

Und die Antwort kommt von einem der tiefen und bis heute noch schwierigen Probleme in der rechnerischen Zahlentheorie.

10.2.1 Der grundlegende RSA-Algorithmus

Wählen wir zwei Primzahlen, p und q , jeweils 1024 Bit, sagen wir, und betrachten das 2048-Bit-Produkt $N = pq$. Wir können einen 2048-Bit-Block einer Nachricht als eine ganze Zahl M modulo 2^{2048} betrachten. Wenn die Nachricht in 8-Bit-ASCII vorliegt, zum Beispiel, dann können die 2048-Bit-Blöcke als eine Ganzzahlbasis $2^8 = 256$ von $2048/8 = 256$ Ziffern betrachtet werden. Wir wählen einen Exponenten e , eine ganze Zahl modulo 2^{2048} , als den öffentlich bekannten Verschlüsselungsexponenten. Wir veröffentlichen e für das bekannte Universum, und wir erlauben daher jedem, uns den Wert $E = M^e \pmod{N}$ zu senden; dies ist die Verschlüsselung der Nachricht M .

Wir kennen die Werte von p und q , haben sie aber geheim gehalten. Wir wissen, dass die Ordnung der Gruppe modulo N $\phi(N) = (p-1)(q-1)$ ist, aber niemand sonst weiß das, weil niemand sonst die Faktorisierung von N in p und q hat, und weil das Berechnen von $\phi(N)$ ohne Faktorisierung von N nach aktuellem Wissensstand ein rechnerisch schwieriges Problem ist.

Da die ganzzahligen Reste modulo N eine Gruppe modulo N der Ordnung $\phi(N)$ bilden, und da wir den Wert von $\phi(N)$ kennen, wissen wir, dass wir einen Wert d bestimmen können, so dass $ed \equiv 1 \pmod{\phi(N)}$. Da $\phi(N)$ die Ordnung der Gruppe modulo der zusammengesetzten Zahl N ist, wissen wir, dass $x^{ed} \equiv x^1 \pmod{N}$ für jedes x gilt.

Daher, ...

$$E^d \equiv (M^e)^d \equiv M^{ed} \equiv M^1 \equiv M \pmod{N}$$

Wir veröffentlichen das Produkt N und den Verschlüsselungsexponenten e . Wir halten die Faktorisierung von N in das Produkt $N = pq$, den Wert von $\phi(N)$, und den Entschlüsselungsexponenten d geheim. Jeder, der uns eine sichere Nachricht M senden möchte, kann $M^e \equiv E \pmod{N}$ senden. Nur wir haben den Wert von d und können entschlüsseln.

Dies ist das grundlegende RSA-Verschlüsselungsverfahren, das für seine Sicherheit auf die Schwierigkeit setzt, d nur mit e und N zu bestimmen.

Die aktuelle Theorie besagt, dass es keinen guten Rechenmechanismus gibt, um d aus e und N zu ermitteln, ohne $\phi(N)$ zu kennen, und dass es keinen guten Rechenmechanismus gibt, um $\phi(N)$ zu kennen, ohne N in p mal q zu zerlegen.

Die aktuelle Theorie besagt also, dass das Knacken des RSA-Algorithmus die Fähigkeit erfordert, große Ganzzahlen (sagen wir, 2048 Bits) zu faktorisieren, und die aktuelle Theorie und Praxis besagen, dass dies im Allgemeinen ein rechnerisch unpraktikables Problem ist.

10.3 Implementierung

Um ein RSA-Verschlüsselungsschema zu implementieren, müssen wir zunächst große Primzahlen p und q finden, jede von (sagen wir) 1024 Bits. Das ist tatsächlich nicht allzu schwierig. Der Primzahlsatz besagt, dass die Anzahl der Primzahlen kleiner oder gleich x ungefähr

$$\frac{x}{\log x}.$$

ist. Das bedeutet, dass es ungefähr

$$2^{1024}/1024$$

Primzahlen von 1024 Bits oder weniger gibt, und ungefähr

$$2^{1023}/1023$$

Primzahlen von 1023 Bits oder weniger, so dass es nach einer groben Zählung fast genauso viele 1024-Bit-Primzahlen gibt wie Primzahlen von weniger als oder gleich 1023 Bits. Primzahlen zu finden ist nicht so schwer. Der Primzahlsatz ist ein asymptotisches Ergebnis, und es gibt Fragen, welche Logarithmen man verwenden könnte, aber 2^{1023} ist eine Zahl von mehr als 300 Dezimalstellen, und wir müssen uns keine Sorgen machen, einen Faktor von 10 oder sogar 100 bei der Schätzung der Rechenkosten zu gewinnen oder zu verlieren.

(Wir werden in den Kap. 11 und 12 sehen, dass es bestimmte Arten von Primzahlen zu vermeiden gilt, aber das ist leicht zu bestimmen, und die Primzahlen, die schlechte Wahlmöglichkeiten sind, können leicht vermieden werden.)

Wir müssen e bestimmen und dann d . Wie bei der Wahl der Primzahlen gibt es bestimmte Werte von e zu vermeiden, aber es gibt viele geeignete e zur Verwendung. Gegeben e , ist die Bestimmung von d nicht schwieriger als zwei Anwendungen eines verallgemeinerten euklidischen Algorithmus, gefolgt von einer Anwendung des chinesischen Restsatzes. Der schwierige Teil beim Aufsetzen eines RSA-Kryptosystems ist also die Bestimmung geeigneter p , q und dann e .

Gegeben das, können wir N und e veröffentlichen und auf Nachrichten warten, die uns mit dem Kryptosystem gesendet werden.

10.3.1 Ein Beispiel

Wir werden das RSA-Kryptosystem anhand eines Beispiels veranschaulichen. Wie aus den Kap. 3 und 4 abgeleitet werden kann, ist die Ordnung der Gruppe modulo $N = pq$ $\phi(N) = (p-1)(q-1)$. Wenn wir p und q so wählen würden, dass sowohl $(p-1)/2$ als auch $(q-1)/2$ prim wären, hätten wir den längstmöglichen multiplikativen Zyklus und

die geringste Anzahl von Nullteiler für jede Gruppe der Größe etwa N , für die N genau zwei Primfaktoren hätte.

Wir werden wählen

$$p = 4294900427 = 2 \times 2147450213 + 1$$

und

$$q = 4294901243 = 2 \times 2147450621 + 1.$$

2147450213 und 2147450621 sind beide Primzahlen. Diese sind zufällig zwei solche Primzahlen mit jeweils 32 Bit, so dass das Produkt

$$N = 18446173182483530761$$

64 Bit lang ist, und

$$\phi(N) = (4294900427 - 1) * (4294901243 - 1) = 18446173173893729092.$$

Wir benötigen einen Verschlüsselungsexponenten e und einen Entschlüsselungsexponenten d , die so gewählt sind, dass

$$ed \equiv 1 \pmod{\phi(N)}.$$

Wenn wir $e = 111111111$ wählen, zeigt ein einfacher euklidischer Algorithmus, dass der Entschlüsselungsexponent $d = 13522443910346794455$ ist und dass

$$111111111 * 13522443910346794455 \equiv 1 \pmod{18446173173893729092}.$$

Wenn unsere zu verschlüsselnde Nachricht die acht Byte lange Nachricht „the text“ ist, wird diese zu

$$\begin{array}{ccccccc} t & h & e & & t & e & x & t \\ 74 & 68 & 65 & 32 & 74 & 65 & 78 & 74 \end{array}$$

wobei die zweite Zeile der ASCII-Code in Hexadezimal ist, und somit ist die Nachricht, in ASCII, deren Bits als 64-Bit-Integer interpretiert werden, der Integer

$$8392569455039047796.$$

Wir verschlüsseln nun, berechnen

$$8392569455039047796^{111111111} \pmod{N} \equiv 2134423211333931089$$

und dann (Wunder der Wunder) entschlüsseln wir zur Kontrolle:

$$2134423211333931089^{13522443910346794455} \pmod{N} \equiv 8392569455039047796.$$

Offensichtlich können wir bei längeren Nachrichten die Nachricht in acht Byte große Blöcke unterteilen und jeden Block separat verschlüsseln. Für eine ernsthafte Implementierung von RSA könnten wir p und q jeweils 1024 Bit lang wählen, für N von 2048 Bit, und dann in 256-Byte-Blöcken verschlüsseln.

10.4 Wie schwer ist es, RSA zu knacken?

Offensichtlich ist RSA nur sicher, wenn es rechnerisch unpraktikabel ist, den Entschlüsselungsexponenten d zu bestimmen, gegeben den Modulus N und den Verschlüsselungsexponenten e . Der aktuelle Stand der Theorie besagt, dass es keine Möglichkeit gibt, d aus e zu berechnen, ohne $\phi(N)$ zu kennen, und dass es keine Möglichkeit gibt, den Wert von $\phi(N)$ zu kennen, ohne N zu faktorisieren. Die Sicherheit von RSA beruht also auf der Schwierigkeit, große Ganzzahlen N zu faktorisieren, wobei N 2048 Bit lang sein könnte und das Produkt von zwei etwa gleich großen Primzahlen p und q ist.

Es gibt sicherlich schlechte Wahlmöglichkeiten für p und q , wie wir in den nächsten Kapiteln sehen werden. Es gibt auch schlechte Wahlmöglichkeiten für e . Aber wie oben erwähnt, gibt es viele Primzahlen, und gute auszuwählen ist wirklich keine schwierige Aufgabe. Darüber hinaus war die Faktorisierung schon vor ihrer kryptographischen Bedeutung ein schwieriges rechnerisches Problem und ist es auch mit dem enormen Interesse, das mit ihrer kryptographischen Bedeutung einhergeht, geblieben. Die jüngste Geschichte der Faktorisierungsrekorde enthält die folgenden in Tab. 10.1 aus einer

Tab. 10.1 Rekordfaktorisierungen aus den RSA-Herausforderungszahlen

Name	Dezimalzahlen	Bits	Faktorisierung bekannt gegeben
RSA-129	129	426	April 1994
RSA-130	130	430	April 1996
RSA-140	140	463	Februar 1999
RSA-150	150	496	2004
RSA-160	160	530	April 2003
RSA-170	170	563	Dezember 2009
RSA-576	174	576	Dezember 2003
RSA-180	180	596	Mai 2010
RSA-190	190	629	November 2010
RSA-640	193	640	November 2005
RSA-200	200	633	Mai 2005
RSA-210	210	696	September 2013
RSA-704	212	704	Juli 2012
RSA-220	220	729	Mai 2016
RSA-230	230	762	August 2018
RSA-232	232	768	Februar 2020
RSA-768	232	768	Dezember 2009
RSA-240	240	795	November 2019
RSA-250	250	829	Februar 2020

Liste von Herausforderungszahlen, die von Rivest, Shamir und Adleman veröffentlicht wurden. In vielen Fällen dauerten die Berechnungen, die zur Faktorisierung dieser Herausforderungswerte benötigt wurden, Monate oder Jahre.

10.5 Andere Gruppen

Es sollte klar sein, dass es keine wirkliche Magie in der Wahl der Ganzzahlfaktorisierung für diese Art von asymmetrischer Kryptographie gibt, bei der das Wissen, wie man eine Nachricht verschlüsselt, nicht impliziert, dass man weiß, wie man die Nachrichten eines anderen entschlüsselt. Der Schlüssel zur Verwendung von RSA als kryptographischem Algorithmus besteht darin, dass man eine Gruppe modulo N hat und dass es, gegeben einen öffentlichen Wert e , der zum Verschlüsseln verwendet wird, schwierig ist, den inversen Wert d in der Gruppe zu bestimmen, weil es schwierig ist, die Ordnung $\phi(N)$ der Gruppe zu bestimmen. Wir müssen einen großen Zyklus in der Gruppenordnung haben (daher unsere Wahl, im Beispiel, der Primzahlen p und q , für die $p - 1$ und $q - 1$ zweimal eine Primzahl waren), um Brute-Force-Angriffe unpraktikabel zu machen.

Wir erinnern daran, dass Fermats kleiner Satz, auf dem die Bestimmung von e und d basiert, eigentlich Lagranges Satz ist, der auf jede Gruppe anwendbar ist. Andere Gruppen könnten verwendet werden und wurden in der Literatur vorgeschlagen. Die wichtigste davon ist die Gruppe der Punkte auf einer elliptischen Kurve, und Kryptographie mit elliptischen Kurven ist das Thema von Kap. 14. Die beiden größten Vorteile von elliptischen Kurven liegen in der Möglichkeit, den gleichen Grad an rechnerischer Unmöglichkeit mit weit weniger Bits (und damit weniger Rechenleistung) zu erreichen und in der Abwesenheit selbst der Art von Angriff, die das General Number Field Sieve für die Faktorisierung ist.

Abgesehen von elliptischen Kurven haben die Vorschläge zur Verwendung anderer Gruppen, obwohl kryptographisch ebenso sicher, sich nicht als vielversprechende Alternativen erwiesen. Im Kern einer RSA-ähnlichen Methode steht die Exponentiation eines Elements in einer Gruppe. Bei RSA ist diese Exponentiation eine modulare Multiplikation, um a^b in einer Gruppe zu berechnen. Für die anderen vorgeschlagenen Gruppen ist die grundlegende Gruppenoperation viel komplizierter und damit viel langsamer, daher die mangelnde Akzeptanz solcher Ideen. Praktisch alle anderen gruppenbasierten Vorschläge erfordern modulare Arithmetik modulo großer Zahlen, und es ist im Grunde unmöglich, die gleiche Mathematik mit weniger Rechenoperationen als RSA zu machen. Ein Quadrat für jedes Bit von N , und eine Multiplikation für die (ungefähr) Hälfte der Bits, die zufällig 1 sind, ist so einfach wie man sich vorstellen kann.

Andererseits ist RSA als kryptographischer Algorithmus langsamer als beispielsweise AES, und RSA *per se* wird nicht umfangreich für Kryptographie verwendet. Eine der Hauptmerkmale von AES ist, dass es byteorientiert ist und tabellengesteuert gemacht werden kann, während die Exponentiation modulo N eine Multiplikation mit mehrpräzisen Ganzzahlen erfordert und von Natur aus langsam ist. Aus diesem Grund, wie

wir in Kap. 13 sehen werden, obwohl RSA nicht weit verbreitet für Kryptographie verwendet wird, wird eine Variante, die auf weitgehend der gleichen zugrunde liegenden Zahlentheorie basiert, für den Schlüsselaustausch verwendet.

10.6 Übungen

1. (Wahrscheinliche Programmieraufgabe.) Verwenden Sie $N = 193 \times 223 = 43039$ als Modulus für ein RSA-Kryptosystem und $e = 23$. Berechnen Sie d . Verschlüsseln und entschlüsseln Sie dann (zu Überprüfungszwecken) die Nachricht „message“ byte für byte (füllen Sie das letzte mit einem Leerzeichen auf). Beachten Sie, dass da 43039 kleiner als 46340 ist, kann die gesamte Arithmetik mit 32-Bit-Signatararithmetik durchgeführt werden, fast unabhängig von der verwendeten Programmiersprache.
2. (Programmieraufgabe.) Verwenden Sie eine Programmiersprache, die Mehrpräzisionsarithmetik zulässt, entweder nativ (Python) oder mit Paketen (C++ oder Java), erweitern Sie Ihre Lösung zur vorherigen Aufgabe, um die Berechnung des Beispiels aus Abschn. 10.3.1 durchzuführen.
3. (Wahrscheinliche Programmieraufgabe.) Es ist verlockend, zwei Primzahlen p und q von jeweils 1024 Bits zu wählen, sagen wir, und dann die Nachricht in $2048/8 = 256$ Blöcke zu zerlegen. Dabei kann jedoch ein Problem auftreten, da der Modulus N kleiner als 2^{2048} sein wird und kleiner als die Ganzzahl sein könnte, die die Nachricht kodiert. Testen Sie dies mit $p = 241$ und $q = 251$, so $N = 60491$, und überprüfen Sie, dass eine zweibytige Nachricht, die zu einer Ganzzahl zwischen 60492 und 65536 kodiert, bei der Entschlüsselung zu einer falschen Nachricht führt.

Literatur

1. W. Diffie, M.E. Hellman, New directions in cryptography. IEEE Trans. Inf. Theory IT-22, 644–654 (1976)
2. R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 120–126 (1978)
3. J.H. Ellis, The history of non-secret encryption. Cryptologia 23, 267–273 (1999)

Wie man eine Zahl faktorisiert

11

Zusammenfassung

Die Sicherheit des RSA-Kryptosystems basiert auf der Schwierigkeit, ganze Zahlen N zu faktorisieren, die das Produkt von zwei großen Primzahlen p und q sind. Wenn p und q gut gewählt sind, dann ist das Faktorisieren von N tatsächlich schwierig, aber es gibt auch Faktorisierungsmethoden, die bei bestimmten Arten von Zahlen sehr schnell arbeiten. Um die Sicherheit eines RSA-Systems zu gewährleisten, muss man sorgfältig ein N wählen, das nicht einer der schnelleren Methoden zum Opfer fällt. Wir werden zuerst die Pollard-Rho- und Pollard- $p-1$ Methoden diskutieren. Diese werden nicht nur allgemein zur Faktorisierung verwendet, sondern wurden auch verallgemeinert, um in anderen Angriffen gegen kryptographische Systeme anwendbar zu sein. Danach gehen wir zu CFRAC über, einem Vorläufer der modernsten Faktorisierungsmethode, die das Hauptthema von Kap. 12 ist.

Widmung

Wir widmen dieses Kapitel dem Andenken an Richard Guy, dessen Arbeit mit demselben Titel [1] wie dieses Kapitel einen legendären Beitrag zur Literatur über Faktorisierung (und somit zur Kryptographie) darstellt. Richard Guy starb im Alter von 103 Jahren am 9. März 2020 und trug bis wenige Wochen vor seinem Tod zur Mathematik bei. Wir hoffen, dass dieses Kapitel einem großen Gelehrten und Freund gerecht wird.

Einleitung

Im Allgemeinen werden wir uns die RSA-Verschlüsselung mit einem Modulus $N = pq$ ansehen, wobei p und q große Primzahlen sind (sagen wir, jeweils 2048 Bit). Es besteht allgemeiner Konsens, dass selbst die leistungsfähigsten Faktorisierungsalgorithmen bei Moduli N von 4096 Bit mit gut gewählten Faktoren p und q nicht erfolgreich sein werden. Es gibt jedoch eine Reihe von Faktorisierungsalgorithmen, die relativ

kostengünstig zu betreiben sind und eine Faktorisierung von N liefern, wenn p und q schlecht gewählt sind, und es gibt mittelgradige Algorithmen, die ebenfalls manchmal erfolgreich sind. Es liegt in der Verantwortung jedes Einzelnen, der die RSA-Verschlüsselung implementieren möchte, N mit diesen geringeren Algorithmen zu testen, um sicherzustellen, dass gute Primzahlen verwendet wurden, oder zumindest Primzahlen zu vermeiden, die wahrscheinlich zu einer kostengünstigeren Faktorisierung von N führen.

Einige der unten beschriebenen Faktorisierungsmethoden sind Heuristiken und werden nicht immer schnell einen Faktor liefern. Einige werden im Grunde immer eine Faktorisierung liefern, aber mit einer Laufzeit, die nachweislich zu lang ist, um bei einer gut gewählten RSA-Implementierung nützlich zu sein.

Wir bemerken zu Beginn, dass mindestens ein Faktor einer beliebigen Ganzzahl N kleiner sein muss als die Quadratwurzel von N , aber dass die Teilung von N durch alle Primzahlen kleiner als \sqrt{N} so unpraktikabel ist, dass sie von vornherein als Möglichkeit ausgeschlossen wird. Der Primzahlsatz [2] besagt, dass die Anzahl der Primzahlen kleiner oder gleich x , die wir als $\pi(x)$ schreiben, erfüllt

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{\frac{x}{\ln x}} = 1$$

und für Werte von x , die so groß sind, wie sie in der RSA-Verschlüsselung auftreten, wird die Grenze von 1 fast erreicht. Das heißt, es gibt, höchstens um einen Faktor, der nicht viel von 1 abweicht, mehr als 10^{600} Primzahlen von 2048 Bit. Wenn wir also $N = pq$ mit p und q jeweils von 2048 Bit annehmen, müssten wir eine Teilung durch mehr als 10^{600} Primzahlen durchführen, was einfach unmöglich ist; der Planet Erde existiert erst seit etwa $4.5 \cdot 10^{18}$ Nanosekunden.

Die Faktorisierung war ein kurioses akademisches Unterfangen vor der Veröffentlichung des RSA-Algorithmus. Viele Jahre lang war die Standardreferenz für „schwierige“ Faktorisierungen, die versucht werden sollten, die „Cunningham-Tabellen“, die erstmals von Allan J. C. Cunningham im Jahr 1925 erstellt und dann von einer Reihe von Autoren aktualisiert wurden [3]. Es gibt eine umfangreiche Literatur aus der Zeit vor der Erfindung des Number Field Sieve (der NFS, den wir fast in Kap. 12) [4–19]. Seit der Einführung des NFS wurde die meiste Arbeit zur Faktorisierung darauf verwendet, die Laufzeit zu verbessern, aber nicht den grundlegenden Algorithmus.

11.1 Pollard Rho

Die Pollard rho Faktorisierungsmethode [1, 20] ist eine Heuristik, die nur für bestimmte Ganzzahlen funktioniert, die man faktorisieren möchte. Es ist jedoch eine einfache und schnelle Methode und kann immer ausprobiert werden, um zu sehen, ob sie funktioniert.

Nehmen wir an, wir versuchen $N = 1037 = 17 \cdot 61$ zu faktorisieren.

Wir iterieren eine einfache Funktion, wie $x_n^2 = x_{n-1}^2 - 1$ wiederholt. Wir beginnen mit x_1 und x_2 , und dann iterieren wir x_1 zu x_2 und iterieren x_2 zweimal, um x_3 zu erhalten und dann x_4 . Wir iterieren dann x_2 einmal und x_4 zweimal. Im stabilen Zustand führen wir drei Iterationen der Funktion durch, um die Werte von x_m und x_{2m} , alle modulo $N = 1037$, zu erhalten. Während wir weitermachen, nehmen wir $\gcd(x_m - x_{2m}, N)$ und hoffen, dass ein Faktor von N herauskommt.

m	x_m	x_{2m}	$x_m - x_{2m}$	ggT
1	2	8	-6	1
2	3	63	-60	1
3	8	252	-244	61
4	63	369	-306	17
5	857			
6	252			
7	246			
8	369			

Was passiert ist folgendes: Wenn man die x_i mit einfacher Geschwindigkeit und doppelter Geschwindigkeit ausführt, müssen die Werte x_i und x_{2i} schließlich kollidieren, da es nur endlich viele Werte modulo p für jeden Primfaktor p von N gibt. Das „rho“ kommt von der Erscheinung des Zyklus, und wir verwenden den Begriff *Epakte* für die Primzahl p für das kleinste m , so dass wir $x_{2m} \equiv x_m \pmod{p}$ haben und m nicht kleiner ist als die Schwanzlänge, die in den Zyklus führt. Wenn N das Produkt von zwei Primzahlen ist, hoffen wir, dass die Epakten für die beiden Faktoren nicht gleich sind, sonst würden wir N für den ggT bekommen und nicht nur einen der beiden Faktoren. In der Praxis wird dies bei großen Primzahlen kein Problem sein.

Und im Allgemeinen würden wir anstatt den ggT bei jedem Schritt zu nehmen, ein laufendes Produkt modulo N ansammeln und nur alle 100 Schritte, oder vielleicht alle 1000 Schritte, je nach Geschmack, ggT's nehmen.

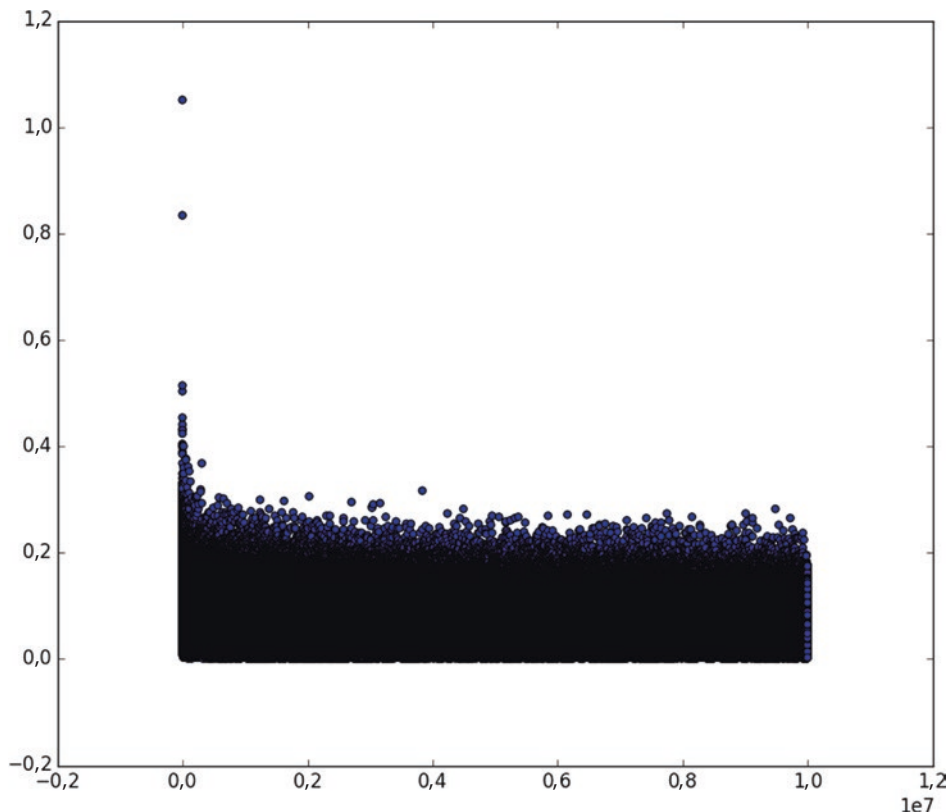
Schließlich präsentieren wir ein Streudiagramm aller Epakten für Primzahlen bis zehn Millionen. Das Streudiagramm ist eigentlich von

$$\frac{epact(p)}{\sqrt{p} \ln p}$$

welches als langsam abnehmende Konstante gesehen werden kann, die im Allgemeinen kleiner als 0,2 für Primzahlen p dieser Größe ist.

Wir stellen jedoch fest, dass diese Methode fast sicher keine Ganzzahlen faktorisieren wird, die für RSA-Verschlüsselung verwendet werden würden. Wenn $N = pq$ und die Primzahlen p und q jeweils 2048 Bit lang sind, dann würden wir erwarten, dass wir

die Iteration mehr als $2^{1024} \approx 10^{308}$ Mal durchlaufen müssen, bevor wir auf die Epakte stoßen und einen Faktor finden.



11.2 Pollard $p - 1$

Die Pollard $p - 1$ Faktorisierungsmethode Methode [21] ist der Vorläufer einer Reihe von Methoden.

Wir erinnern uns an den Satz von Lagrange, dass die Ordnung der multiplikativen Gruppe mod einer Primzahl p ist $p - 1$ und dass jedes Element, das zur Ordnung der Gruppe erhoben wird, die Identität ist.

Um eine ganze Zahl N zu faktorisieren, berechnen wir zuerst einen enormen Exponenten $M = \prod q_i^{e_i}$, der das Produkt aller kleinen Primzahlen q_i zu hohen Potenzen e_i ist, für einige vage Definitionen von „alle“, „klein“ und „hoch“. Zum Beispiel könnte man die 78498 Primzahlen bis zu einer Million zur höchsten Exponenten nehmen, so

dass $q_i^{e_i}$ immer noch eine 32-Bit-Zahl ist. Dieser Wert von M ist eine ganze Zahl von etwa 3 Millionen Bits.

Wir wählen jetzt einen Rest a (wie 3) und berechnen

$$a^M \equiv b \pmod{N}.$$

Jetzt, wenn es passiert, dass N durch eine Primzahl p teilbar ist, für die $p - 1 \mid M$ teilt, dann haben wir

$$a^M \equiv b \equiv 1 \pmod{p}$$

und durch die Einnahme von

$$\gcd(b - 1, N)$$

extrahieren wir den Faktor p .

Im Allgemeinen würden wir nicht erwarten, dass wir in der Lage sind, RSA-artige N mit diesem Ansatz zu faktorisieren. Damit die $p - 1$ Methode funktioniert, müsste die Ordnung der Gruppe modulo einiger Faktor von N „krümelig“ sein, das heißt, sie müsste nur aus kleinen Primfaktoren bestehen, und das ist unwahrscheinlich. Die Existenz dieses Faktorisierungsangriffs bedeutet jedoch, dass man speziell davor schützen sollte, ihm ausgesetzt zu sein, wenn man die beiden Primzahlen auswählt, mit denen man N erstellt. Ein Teil der Auswahl der Primzahlen p und q zur Erstellung von $N = pq$ muss darin bestehen, zu überprüfen, dass jeder von $p - 1$ und $q - 1$ einen sehr großen Primfaktor hat; das würde einen Pollard $p - 1$ Angriff verhindern.

11.2.1 Die allgemeine Metaphysik von $p - 1$

Die $p - 1$ Methode ist der Vorläufer einer Reihe solcher Algorithmen.

Wir nehmen an, dass wir eine Gruppe haben, wie die Gruppe der Reste unter Multiplikation modulo einer Primzahl. Wir nehmen an, dass wir eine Möglichkeit haben, die Ordnung der Gruppe zu einem Faktor von N zu „hacken“; im Falle des traditionellen $p - 1$, das ist der $\gcd(b - 1, N)$ Teil. Die gesamte Methode ist ein bisschen ein Hack im ursprünglichen Sinne des Begriffs. Aber es funktioniert.

Also potenzieren wir einfach ein Gruppenelement zu einer großen Potenz M in der Hoffnung, dass der Faktor aus der Gruppenstruktur hervortritt.

Dieser grundlegende Ansatz zeigt sich später in anderen Faktorisierungsmethoden, die verschiedene Gruppen anstelle der reinen Zahlen modulo N verwenden.

11.2.2 Schritt Zwei von $p - 1$

Was oben beschrieben wurde, ist Schritt Eins der $p - 1$ Methode. Schritt Zwei ist wie folgt:

Jenseits eines bestimmten Punktes würde man nicht erwarten, dass $p - 1$, die Ordnung der Gruppe, durch das Quadrat einer Primzahl teilbar ist, so dass wir in M nur Primzahlen zur ersten Potenz einbeziehen müssen. Und jenseits eines bestimmten Punktes könnten wir annehmen, dass uns nur eine einzige Primzahl in der Ordnung der Gruppe fehlt, das heißt, dass M alle $p - 1$ enthält, bis auf eine letzte Primzahl, die etwas größer ist als die Primzahlen, die wir in unsere Berechnung von M einbezogen haben. Also potenzieren wir eine weitere Primzahl nach der anderen.

Wenn wir ursprünglich gewählt haben, die Primzahlen bis 100 Tausend zu nehmen, dann wären die nächsten Primzahlen 100003, 100019, 100043, 100049 und 100057. Wir potenzieren zur 100003-ten Potenz, um 100003 zu M hinzuzufügen, und nehmen entweder den ggT oder multiplizieren den $b - 1$ Wert in ein laufendes Produkt ein, genau wie wir es mit Pollard rho gemacht haben. Dann potenzieren wir nur noch 16 weitere Schritte, um 100003 durch 100019 zu ersetzen, dann 24 weitere Schritte, um 100019 durch 100043 zu ersetzen, und so weiter. Das ist günstig und kann effektiv sein.

11.3 CFRAC

CFRAC, der fortgesetzte Bruch Algorithmus, wurde erstmals von Lehmer und Powers um 1930 vorgeschlagen, aber erst 1970 ernsthaft implementiert, weil er nicht immer funktioniert. Das macht es von Hand mühsam ärgerlich, aber weniger so, wenn es der Computer ist, der einfach mehrere Male scheitert, bevor er Erfolg hat. Tatsächlich begann die Verwendung von Morrison und Brillhart [22, 23] im Jahr 1970 zur Faktorisierung von $F_7 = 2^{2^7} + 1$, einer Ganzzahl von 39 Dezimalstellen, eine völlig neue Ära in der Forschung zur Faktorisierung.

Mersenne hat bereits 1643 die Methode der Quadratdifferenzen zur Faktorisierung von Ganzzahlen verwendet. Wenn eine zu faktorisierende Ganzzahl N als Differenz von Quadraten geschrieben werden kann, sagen wir

$$N = x^2 - y^2, \quad (11.1)$$

dann haben wir eine algebraische Faktorisierung

$$N = (x + y)(x - y).$$

das Problem ist natürlich, dass dies für große N unpraktisch ist, da es nicht viel gibt, das getestet werden kann und viel besser aussieht als die Probedivision.

CFRAC und die Siebmethoden, die ihm gefolgt sind und die nun die Arbeitspferdmethoden zum Faktorisieren großer N sind, basieren auf einer Vereinfachung der obigen Gleichung: Anstatt Gl. (11.1) als eine Gleichung zu lösen, lösen wir die Kongruenz

$$x^2 - y^2 \equiv 0 \pmod{N}$$

und dann berechnen wir

$$m = \gcd(N, x + y)$$

und

$$n = \gcd(N, x - y).$$

Wir haben eine algebraische Faktorisierung von N in $x + y$ mal $x - y$, und wir hoffen, dass wenn $N = pq$ für ein kryptographisches N , das das Produkt von zwei großen Primzahlen p und q ist, wir nicht die trivialen Faktorisierungen $(m, n) = (1, N)$ oder $(m, n) = (N, 1)$ erhalten, wenn wir die beiden ggT berechnen. Wenn wir eines dieser beiden Ergebnisse erhalten, suchen wir weiter nach einem anderen Paar von x und y Werten, die verwendet werden sollen. (Dies ist der Fehlerpart, der es mühsam machen würde, ihn von Hand zu berechnen.)

11.3.1 Fortgesetzte Brüche

Wir veranschaulichen das Konzept einer Kettenbruch durch ein Beispiel. Betrachten Sie den Wert $267/111$. Wir schreiben dies als

$$267/111 = 2 + 45/111$$

Der Algorithmus sollte offensichtlich sein (obwohl es nicht unbedingt klar sein mag, warum jemand einen solchen Algorithmus implementieren möchte):

- Schreiben Sie die Menge als $a_0 + z_0$, wo $0 \leq z_0 < 1$.
- Schreiben Sie z_0 als $\frac{1}{1/z_0}$, beachten Sie, dass $1/z_0 > 1$.
- Schreiben Sie $\frac{1}{1/z_0}$ als $\frac{1}{a_1 + z_1}$, wo $0 \leq z_1 < 1$.
- Spülen und wiederholen.

Lassen Sie uns die folgenden Wiederholungen mit etwas motivieren, das nicht ganz ein Beweis ist, aber nahe kommt und die Wiederholungen verständlicher machen sollte.

Nehmen wir an, wir haben einen Kettenbruch

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}}$$

den wir als

$$R = [a_0, a_1, a_2, a_3, \dots]$$

für Ganzzahlen a_i schreiben, und für den Moment kümmern wir uns nicht darum, ob es sich um einen endlichen oder unendlichen Kettenbruch handelt.

Wir werden bewerten

$$R_0 = p_0/q_0 = [a_0]$$

$$R_1 = p_1/q_1 = [a_0, a_1]$$

$$R_2 = p_2/q_2 = [a_0, a_1, a_2]$$

$$R_3 = p_3/q_3 = [a_0, a_1, a_2, a_3]$$

Dies sind alles rationale Zahlen, da die a_i Ganzzahlen sind, und somit können wir annehmen, dass p_i und q_i Ganzzahlen sind. Wir werden p_i und q_i die *Konvergenten* nennen.

Lassen Sie uns also die Algebra entwirren.

$$R_0 = p_0/q_0 = [a_0]$$

so haben wir

$$p_0 = a_0$$

$$q_0 = 1$$

Nun

$$R_1 = p_1/q_1 = a_0 + \frac{1}{a_1} = \frac{a_0 a_1 + 1}{a_1}$$

so haben wir

$$p_1 = a_0 a_1 + 1 = a_1(p_0) + 1$$

$$q_1 = a_1 = a_1(q_0) + 0$$

Weiterhin haben wir

$$\begin{aligned} R_2 = p_2/q_2 &= a_0 + \frac{1}{a_1 + \frac{1}{a_2}} \\ &= a_0 + \frac{1}{\frac{a_1 a_2 + 1}{a_2}} \\ &= a_0 + \frac{a_2}{a_1 a_2 + 1} \\ &= \frac{a_0 a_1 a_2 + a_0 + a_2}{a_1 a_2 + 1} \\ &= \frac{a_2(a_0 a_1 + 1) + a_0}{a_2(a_1) + 1} \\ &= \frac{a_2(p_1) + p_0}{a_2(q_1) + q_0} \end{aligned}$$

so haben wir

$$\begin{aligned} p_2 &= a_2(a_0a_1 + 1) + a_0 = a_2(p_1) + p_0 \\ q_2 &= a_2(a_1) = a_2(q_1) + q_0 \end{aligned}$$

Schließlich

$$\begin{aligned} R_3 &= p_3/q_3 = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3}}} \\ &= a_0 + \frac{1}{a_1 + \frac{1}{\frac{a_2a_3+1}{a_3}}} \\ &= a_0 + \frac{1}{a_1 + \frac{a_3}{a_2a_3+1}} \\ &= a_0 + \frac{1}{\frac{a_1a_2a_3+a_1+a_3}{a_2a_3+1}} \\ &= a_0 + \frac{a_2a_3+1}{a_1a_2a_3+a_1+a_3} \\ &= \frac{a_0a_1a_2a_3+a_0a_1+a_0a_3+a_2a_3+1}{a_1a_2a_3+a_1+a_3} \\ &= \frac{a_3(a_0a_1a_2+a_0+a_2)+a_0a_1+1}{a_3(a_1a_2+1)+a_1} \end{aligned}$$

so haben wir

$$\begin{aligned} p_3 &= a_3(a_0a_1a_2 + a_0 + a_2) + a_0a_1 + 1 = a_3(p_2) + p_1 \\ q_3 &= a_3(a_1a_2 + 1) + a_1 = a_3(q_2) + q_1 \end{aligned}$$

Wir haben den Begriff *konvergent* für die aufeinanderfolgenden Anfangsteile des fortgesetzten Bruchs verwendet. Obwohl wir es nicht beweisen werden, können wir durch Betrachtung der Konvergenten aus unserem Beispiel von $267/111 = 89/37 \approx 2,405$ erkennen, warum dieser Begriff verwendet wird.

$$\begin{aligned} R_0 &= [2] = 2 = 2,0 \\ R_1 &= [2, 2] = 2 + \frac{1}{2} = 2,5 \\ R_2 &= [2, 2, 2] = 2 + \frac{1}{2 + \frac{1}{2}} = \frac{12}{5} = 2,4 \\ R_3 &= [2, 2, 2, 7] = 2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{7}}} = \frac{89}{37} \end{aligned}$$

Wir stellen fest, dass die mit geraden Subskripten versehenen R_i kleiner und die mit ungeraden Subskripten versehenen größer sind als der Wert, dessen fortgesetzten Bruch wir berechnen. Die R_i können als konvergierend gezeigt werden.

Satz 11.1 Wenn r eine rationale Zahl ist, dann hat sie eine endliche fortgesetzte Bruchentwicklung.

Satz 11.2 Wenn $z \in \mathbb{R}$, dann hat sie eine fortgesetzte Bruchentwicklung, die schließlich periodisch ist, wenn und nur wenn sie $a + b\sqrt{n}$ ist, wo $n \in \mathbb{Z}$.

Das zweite dieser Theoreme besagt, dass quadratische Irrationalzahlen fortgesetzte Bruchentwicklungen haben, die schließlich periodisch sind.

11.3.2 Der CFRAC-Algorithmus

Unser Ziel ist es, N zu faktorisieren. Wir erweitern, was im Wesentlichen der fortgesetzte Bruch für \sqrt{N} ist. Was uns motiviert, ist der Gedanke, dass wir, wenn wir eine Kongruenz

$$X^2 \equiv Y^2 \pmod{N},$$

erstellen können, dann hätten wir

$$(X - Y)(X + Y) \equiv 0 \pmod{N},$$

und es könnte passieren, dass einer der beiden Faktoren von N $X - Y$ teilt und der andere $X + Y$. Wenn dies der Fall wäre, dann würde das Berechnen von $\gcd(N, X - Y)$ einen Faktor von N hervorbringen.

Lassen Sie

$$i = 0$$

$$P_0 = 0$$

$$Q_0 = 1$$

$$a_0 = [\sqrt{N}]$$

$$p_{-2} = 0$$

$$p_{-1} = 1$$

$$q_{-2} = 1$$

$$q_{-1} = 0$$

Wir wiederholen nun:

$$\begin{aligned}
 p_i &= a_i \cdot p_{i-1} + p_{i-2} \pmod{N} \\
 q_i &= a_i \cdot q_{i-1} + q_{i-2} \pmod{N} \\
 P_{i+1} &= a_i \cdot Q_i - P_i \\
 Q_{i+1} &= \frac{N - P_{i+1}^2}{Q_i} \\
 a_i &= \left\lfloor \frac{P_{i+1} + \sqrt{N}}{Q_{i+1}} \right\rfloor
 \end{aligned}$$

Wir stellen fest, dass wir immer

$$p_i^2 - Nq_i^2 = (-1)^{i+1} Q_{i+1}. \quad (11.2)$$

Wir versuchen zu erstellen

$$X^2 \equiv Y^2 \pmod{N},$$

und der Quadratwert auf einer Seite, p_i^2 , ist immer vorhanden. Das Wesen aller modernen schweren Faktorisierungsmethoden besteht darin, eine Teilmenge der $(-1)^i Q_i$ zu finden, die, wenn sie zusammen multipliziert wird, ein Quadrat bildet.¹

Um diese Teilmenge von $(-1)^i Q_i$ zu finden, faktorisieren wir die $(-1)^i Q_i$ über eine *Faktorbasis* von „kleinen“ Primzahlen, wobei wir nur die Parität der Exponenten speichern. Wenn wir an einen Punkt gelangen, an dem wir eine Teilmenge von Indizes S haben, so dass

$$\prod_{i \in S} (-1)^i Q_i \pmod{N}$$

eine Quadratzahl ist, dann haben wir eine gewünschte Kongruenz mit

$$X = \prod_{i \in S} p_{i-1} \pmod{N}$$

und

$$Y = \sqrt{\prod_{i \in S} (-1)^i Q_i} \pmod{N}$$

¹Das ist nicht ganz richtig; das Zahlkörpersieb verwendet Wurzeln von Polynomen nicht vom Grad 2, sondern normalerweise vom Grad 5 oder 6, und etwas ausgefeiltere algebraische Zahlentheorie, aber das Wesen des Rechenteils ist das gleiche.

mit Y einer ganzen Zahl, und

$$X^2 \equiv Y^2 \pmod{N}.$$

Es sei denn, wir haben extrem Pech, werden wir feststellen, dass $\gcd(X - Y, N)$ und $\gcd(X + Y, N)$ Faktoren von N liefern werden.

Wir nennen eine ganze Zahl *glatt* wenn sie vollständig über die Faktorbasis faktorisiert. Der CRAC-Algorithmus ist erfolgreich, wenn man genügend glatte Zahlen mit ihren Faktorisierungen erhält, um die linearen Kombinationen zu ermöglichen, die zwei Quadrate erzeugen, die modulo N kongruent zueinander sind. Die Größe der Faktorbasis ist wichtig. Wenn sie zu klein gewählt wird, werden nicht genug der Q_i vollständig faktorisiert. Wenn sie zu groß gewählt wird, dann wird das Testen einzelner Zahlen auf Glätte zu langsam sein. Und was wichtig zu wissen ist, aber was wir nicht beweisen werden, ist, dass die Größenordnungen der Q_i durch \sqrt{N} begrenzt sind; wir haben eine Kontrolle über die Größe der Zahlen, die wir auf Glätte testen.

Und wir bemerken, dass die einzigen Primzahlen, um die wir uns in der Faktorbasis kümmern müssen, diejenigen sind, für die N ein quadratischer Rest ist; für jede Primzahl q , die die Q_i von Gl. (11.2) teilt, müssen wir haben

$$p_i^2 - Nq_i^2 \equiv 0 \pmod{q}.$$

Es besteht immer die Möglichkeit, dass wir Pech haben und einen ggT erhalten, der entweder 1 oder ganz N ist. Dies ist der Punkt, an dem der Algorithmus scheitern kann und der Grund, warum er bei seiner ersten Erfindung nicht verfolgt wurde. Wenn Menschen die Berechnungen durchführen, muss der Erfolg das Ergebnis sein; wenn Computer verwendet werden, kann man einfach das Programm weiterarbeiten lassen, bis der Erfolg schließlich erreicht ist.

11.3.3 Beispiel

Lassen Sie $N = 1000009$ und verwenden Sie als unsere Faktorbasis die Primzahlen 19 und kleiner. (Wir können 7 und 11 überspringen, da N kein quadratischer Rest modulo dieser Primzahlen ist.) Wir schließen -1 als „Primzahl“ ein, da die Vorzeichen auf den Q_i wechseln. Die fett gedruckten Primzahlen sind größer als die größte Primzahl in unserer Faktorbasis, und die entsprechenden Q_i sind nicht glatt.

i	p_{i-1}	q_{i-1}	P_i	Q_i	a_i	Q_i factored
1	1000	1	1000	9	222	$[-1, 3, 3]$
2	222001	222	998	445	4	$[2, 3, \mathbf{37}]$
3	889004	889	782	873	2	$[3, 3, \mathbf{97}]$
4	1000000	2000	964	81	24	$[3, 3, 3, 3]$
5	888788	48889	980	489	4	$[3, \mathbf{163}]$
6	555116	197556	976	97	20	$[\mathbf{97}]$
7	991009	999982	964	729	2	$[-1, 3, 3, 3, 3, 3, 3]$
8	537116	197502	494	1037	1	$[17, \mathbf{61}]$
9	528116	197475	543	680	2	$[-1, 2, 2, 2, 5, 17]$
10	593339	592452	817	489	3	$[3, \mathbf{163}]$
11	308115	974822	650	1181	1	$[\mathbf{1181}]$
12	901454	567265	531	608	2	$[2, 2, 2, 2, 2, 19]$
13	111005	109334	685	873	1	$[3, 3, \mathbf{97}]$
14	12450	676599	188	1105	1	$[5, 13, 17]$
15	123455	785933	917	144	13	$[-1, 2, 2, 2, 2, 3, 3]$
16	617356	893638	955	611	3	$[13, \mathbf{47}]$
17	975514	466820	878	375	5	$[-1, 3, 5, 5, 5]$
18	494881	227711	997	16	124	$[2, 2, 2, 2]$
19	340200	702732	987	1615	1	$[-1, 5, 17, 19]$
20	835081	930443	628	375	4	$[3, 5, 5, 5]$
21	680497	424468	872	639	2	$[3, 3, \mathbf{71}]$
22	196057	779370	406	1307	1	$[\mathbf{1307}]$
23	876554	203829	901	144	13	$[-1, 2, 2, 2, 2, 3, 3]$
24	591160	429120	971	397	4	$[\mathbf{397}]$
25	241167	920300	617	1560	1	$[-1, 2, 2, 2, 3, 5, 13]$

Wir suchen nun nach Teilmengen dieser Liste, bei denen die Faktorisierungen Exponenten haben, die alle zu Null mod 2 addieren (was bedeutet, dass das Produkt der entsprechenden Q_i eine perfekte Quadratzahl wäre). Wir stellen fest, dass wir -1 behandeln, als ob es eine Primzahl wäre; wir benötigen eine gerade Anzahl von -1 Werten in unseren Faktorisierungen, um ein Quadrat (und nicht das Negative eines Quadrats) zu erhalten. Wir stellen fest, dass

$\{4\}$
 $\{18\}$
 $\{1, 7\}$
 $\{1, 15\}$
 $\{1, 23\}$
 $\{1, 17, 20\}$
 $\{9, 12, 19\}$
 $\{9, 14, 20, 25\}$

die Teilmengen sind, die funktionieren.

Wenn wir uns Zeile 18 ansehen, haben wir eine Q_i , die selbst eine perfekte Quadratzahl ist, und somit

$$X = 494881$$

$$Y = 4$$

und wir stellen fest, dass

$$\gcd(494877, N) = 293$$

für eine Faktorisierung gilt.

Im Allgemeinen würden wir nicht erwarten, eine Q_i zu erhalten, die an sich schon ein perfektes Quadrat ist, aber wir könnten uns die Zeilen 1 und 23 anschauen:

$$X = 1000 \cdot 876554 \equiv 546116 \pmod{N}$$

$$Y = 36$$

und wir stellen fest, dass

$$\gcd(546152, N) = 293$$

für eine Faktorisierung gilt.

11.3.4 Berechnung

Bei allen Faktorisierungsmethoden, die tatsächlich funktionieren, gibt es ein Zusammenspiel zwischen der Mathematik und der Berechnung. Im Falle von CFRAC beobachten wir zwei Dinge, die die Berechnung machbar machen. Erstens sind die Q_i , die wir versuchen, vollständig über die Faktorbasis zu faktorisieren, immer kleiner als \sqrt{N} . Das könnte immer noch eine große Zahl sein, aber sie ist zumindest unter Kontrolle.

Gleich wichtig, wenn nicht sogar wichtiger, ist die Bestimmung, welche Q_i multipliziert werden können, um ein Quadrat zu bilden, kann mit Matrixreduktion durchgeführt werden, und zwar mit Matrixreduktion auf Bits und nicht tatsächlich auf Ganzzahlen. Wir erzeugen eine Matrix, in der nur der Exponent modulo 2 der Faktorisierungen vorhanden ist, und wir reduzieren diese Matrix modulo 2. Jede Zeile, die auf lauter Nullen reduziert wird, entspricht einer Teilmenge von Q_i , die eine $X^2 \equiv Y^2 \pmod{N}$ Kongruenz liefert. In unserem obigen Beispiel würden wir mit einer Matrix beginnen

	-1	2	3	5	13	17	19
1	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0
9	1	1	0	1	0	1	0
12	0	1	0	0	0	0	1
14	0	0	0	1	1	1	0
15	1	0	0	0	0	0	0
17	1	0	1	1	0	0	0
18	0	0	0	0	0	0	0
19	1	0	0	1	0	1	1
20	0	0	1	1	0	0	0
23	1	0	0	0	0	0	0
25	1	1	1	1	1	0	0

Tatsächlich wird dies für ein RSA-großes N zu faktorisieren eine große Matrix sein, und wir werden auch die Identitätsmatrix benötigen, während wir reduzieren, um zu bestimmen, welche Zeilen der Matrix Produkte von Q_i erzeugen, die perfekte Quadrate sind, aber die Tatsache, dass wir nur modulo 2 reduzieren müssen, ist ein enormer rechnerischer Vorteil. Wir werden später, im Kontext des diskreten Logarithmusproblems, eine ähnliche Matrix sehen, für die wir jedoch alle ganzzahligen Werte der Matrixreduktion behalten müssen.

11.4 Faktorisierung mit elliptischen Kurven

Die als nächstes zu beschreibenden Siebmethoden sind „mittlere“ Methoden in Kap. 12.

Eine andere mittlere Methode verwendet elliptische Kurven [24], und ist im Wesentlichen eine Anwendung eines Pollard $p - 1$ Ansatzes auf die Gruppe der Punkte auf einer elliptischen Kurve.

Wir erinnern uns, dass in projektiver Notation die Identität der Gruppe modulo einer Primzahl p der Punkt mit projektiven Koordinaten und $z \equiv 0 \pmod{p}$ ist. Wenn wir eine $p - 1$ -ähnliche Multiplikation von Punkten durchführen (beachten Sie, dass die Kurvengruppe additiv geschrieben ist, daher verwenden wir den Begriff „Multiplikation“ anstelle des Begriffs „Exponentiation“), aber die Koordinaten modulo N (die zu faktorisierende Ganzzahl) anstelle des unbekannten Faktors p von N nehmen, dann wird ein ggT der z -Koordinate mit N das p extrahieren, wenn wir die Identität der Gruppe treffen.

In der $p - 1$ Methode ist die Ordnung der Gruppe $p - 1$, und um eine Faktorisierung zu erreichen, benötigen wir einen Exponenten, der alle Faktoren von $p - 1$ enthält. Nach Hasse's Theorem liegt die Ordnung der Kurvengruppe modulo einer Primzahl p im Bereich $p + 1 - 2\sqrt{p}$ bis $p + 1 + 2\sqrt{p}$, so dass wir hier nicht viel schlechter dran sind als mit der naiven $p - 1$. Wir verlieren Rechengeschwindigkeit mit elliptischen Kurven, da die Arithmetik zur Addition von zwei Punkten etwa ein Dutzend Multiplikationen von Ganzzahlen erfordert (die Anzahl variiert je nachdem, welche Kurvenrepräsentation verwendet wird und wie die arithmetischen Schritte angeordnet sind) anstatt nur der einen, die in $p - 1$ verwendet wird.

Jedoch, wir erzielen einen großen Gewinn mit elliptischen Kurven, wie wir es mit dem MPQS von Kap. 12 tun, insofern als wir die Berechnung mit einer großen Anzahl verschiedener Kurven parallel durchführen können. Mit $p - 1$ haben wir nur die eine Gruppenordnung von $p - 1$, die einen großen Primfaktor haben könnte, der die $p - 1$ Methode am Erfolg hindern würde. Mit elliptischen Kurven wissen wir aus der Mathematik, dass im Grunde alle Werte im Bereich $p + 1 - 2\sqrt{p}$ bis $p + 1 + 2\sqrt{p}$ tatsächlich Ordnungen von Kurvengruppen sind, und wir benötigen nur eine dieser Gruppenordnungen, um bröckelig zu sein, um Erfolg zu haben. Dies hat zur Verwendung des ECM als Standardmethode zur Entfernung relativ kleiner Faktoren von beliebigen großen Zahlen, die faktorisiert werden müssen, geführt.

11.5 Übungen

1. Faktorisiere $29 \times 53 = 1537$ mit Pollard Rho.
2. (Programmierübung.) Implementiere eine einfache Version von Pollard $p - 1$ (einfach bedeutet, dass du eine Schleife über die Primzahlpotenzen laufen lassen kannst, die M bilden würden, anstatt einen einzelnen Wert von M zu erstellen). Verwende dies, um 43039 zu faktorisieren.
3. (Programmierübung.) Die Menge der 2×2 Matrizen mit ganzzahligen Koeffizienten, modulo einer ganzen Zahl N , mit der Determinante 1, bilden eine endliche Gruppe. Man könnte daher eine Analogie von Pollard $p - 1$ Faktorisierung durchführen, indem man Potenzen einer solchen Matrix nimmt. Implementiere diesen Algorithmus und verwende ihn, um 1037, 1537 und 43039 zu faktorisieren. Dies ist ein Beispiel für die Metaphysik von $p - 1$ angewendet auf eine andere Gruppe. Was sind die Kosten für jeden Schritt dieses Algorithmus und wie vergleichen sie sich mit dem ursprünglichen $p - 1$?
4. (Programmierübung.) Programmiere den Kettenbruchalgorithmus, verwende die Probedivision, um auf Glätte zu testen, und verwende dies, um die Faktorisierung von 1537 mit einer Faktorbasis von Primzahlen bis 11 einzurichten. Du musst die lineare Algebra nicht codieren, weil du X und Y von Hand berechnen können solltest.

Literatur

1. R.K. Guy, How to factor a number. *Congressus numerantium*, 49–89 (1976)
2. G.H. Hardy, E.M. Wright, *An introduction to the theory of numbers*, 4. Aufl. (Oxford, 1960), S. 223–225
3. J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, S.S. Wagstaff, *Factorizations of $b^n + 1$* , $b = 2, 3, 5, 6, 7, 10, 11, 12$, up to high powers (American Mathematical Society, 1983)
4. R.P. Brent, An improved Monte Carlo factorization algorithm. *BIT*, 176–184 (1980)
5. R.P. Brent, J.M. Pollard, Factorisation of the eighth Fermat number. *Math. Comput.* 627–630 (1981)
6. D.A. Buell, Factoring: algorithms, computers, and computations. *J. Supercomput.* 191–216 (1987)
7. J.P. Buhler, H.W. Lenstra, C. Pomerance, Factoring integers with the number field sieve, in *The development of the number field sieve*, Bd. 1554, Lecture notes in mathematics, Hrsg. von A.K. Lenstra, H.W. Lenstra Jr. (1993), S. 50–94
8. D. Coppersmith, Specialized integer factorization, in *Advances in cryptology – EUROCRYPT '98*, Bd. 1403, Lecture notes in computer science, Hrsg. von K. Nyberg (1998), S. 542–545
9. J.D. Dixon, Asymptotically fast factorization of integers. *Math. Comput.* 255–260 (1981)
10. J.D. Dixon, Factorization and primality tests. *Am. Math. Mon.* 333–352 (1984)
11. J.L. Gerver, Factoring large numbers with a quadratic sieve. *Math. Comput.* 287–294 (1983)
12. J. Cowie, B. Dodson, R.M. Elkenbracht-Huizing, A.K. Lenstra, P.L. Montgomery, J. Zayer, A world wide number field sieve factoring record: on to 512 bits, in *Advances in cryptology – ASIACRYPT '96*, Bd. 1163, Lecture notes in computer science, Hrsg. von K. Kim, T. Matsumoto (1996), S. 382–394
13. H.W. Lenstra Jr., A.K. Lenstra, M.S. Manasse, J.M. Pollard, The factorization of the ninth Fermat number. *Math. Comput.* 319–349 (1993)
14. H.W. Lenstra Jr., C. Pomerance, A rigorous time bound for factoring integers. *J. AMS* 483–516 (1992)
15. C. Pomerance, Analysis and comparison of some integer factoring algorithms, in *Computational methods in number theory*, Math Centre Tracts-Part 1, Hrsg. von H.W. Lenstra Jr., R. Tijdeman (1982), S. 89–139
16. C. Pomerance, The quadratic sieve factoring algorithm, in *Advances in cryptology – EUROCRYPT 1984*, Bd. 209, Lecture notes in computer science, Hrsg. von T. Beth, N. Cot, I. Ingemarsson (1985), S. 169–182
17. S.S. Wagstaff, J. Smith, Methods of factoring large integers, in *Number theory*, Bd. 1240, Lecture Notes in Computer Science, Hrsg. von D.V. Chudnovsky, G.V. Chudnovsky, H. Cohn, M.B. Nathanson (1987), S. 281–303
18. H.C. Williams, A $p+1$ method of factorization. *Math. Comput.* 225–234 (1982)
19. H.C. Williams, An overview of factoring, in *Advances in cryptology*, Hrsg. von D. Chaum (Springer, Boston, 1984), S. 71–80
20. J.M. Pollard, A Monte Carlo method for factorization. *BIT* 331–334 (1975)
21. J.M. Pollard, Theorems on factorization and primality testing. *Math. Proc. Camb. Philos. Soc.* **76**, 521–528 (1974)
22. M.A. Morrison, J. Brillhart, The factorization of F_7 . *Bull. AMS* 264 (1971)
23. M.A. Morrison, J. Brillhart, A method of factoring and the factorization of F_7 . *Math. Comput.* 183–205 (1975)
24. H.W. Lenstra Jr., Factoring integers with elliptic curves. *Ann. Math.* **126**, 649–673 (1987)



Zusammenfassung

In Kap. 11 haben wir mehrere Faktorisierungsmethoden beschrieben. Jede davon wird bei einigen Ganzzahlen erfolgreich sein, aber keine davon ist eine hochmoderne Methode, von der wir erwarten würden, dass sie bei einer gut gewählten RSA $N = pq$ erfolgreich ist. Selbst die beste davon, CFRAC, leidet unter der Notwendigkeit, eine Probedivision durchzuführen, die die meiste Zeit keinen Fortschritt in Richtung Faktorisierung von N erbringt. In diesem Kapitel diskutieren wir Siebmethode zur Faktorisierung. Der primäre rechnerische Vorteil einer Siebmethode besteht darin, dass alle ausgeführten Rechenschritte tatsächlich zur Findung von Faktoren beitragen und dass ein Sieb, das mit konstantem Schritt durch ein Array im Speicher geht, auf den niedrigsten Ebenen eines Rechenprozesses äußerst effizient ist. Wir diskutieren das Quadratische Sieb und das Mehrpolige Quadratische Sieb und schließen dann mit einem Hinweis auf die derzeit beste Methode zur Faktorisierung großer „schwerer“ Ganzzahlen, das Zahlkörpersieb.

12.1 Mängel von CFRAC

Ein großer Vorteil von CFRAC war die Tatsache, dass man zur Faktorisierung einer Ganzzahl N nur Ganzzahlen der Größe \sqrt{N} auf Glätte testen musste.

Der größte Nachteil von CFRAC war, dass es keine Struktur oder Ordnung in den Zahlen gab, die man auf Glätte testete, und somit ist der Glättetest für CFRAC ein Prozess, der im Wesentlichen eine Probedivision ist, die die meiste Zeit scheitert.

12.2 Das Quadratische Sieb

Carl Pomerance erhält normalerweise den größten Teil der Anerkennung für den Algorithmus des quadratischen Siebs (QS), aber seine Wurzeln gehen auf Ideen von Kraitchik zurück. Allerdings war das QS, genau wie CFRAC, eine Idee, die vor der Computerzeit zwar konzipiert, aber nicht vernünftig umzusetzen war. Kraitchik hätte es in seiner Zeit nicht implementiert.

Die Grundidee des QS ähnelt sehr der von CFRAC: Wir betrachten eine lange Liste von Werten Q , für die wir eine Lösung A zur Kongruenz $A^2 \equiv Q$ haben. Wir werden das Q mit einer Faktorbasis faktorisieren. Und dann werden wir die lineare Algebra auf den Faktorisierungen der Q durchführen, um eine Differenz-von-Quadraten-Kongruenz

$$\left(\prod A_i\right)^2 \equiv \prod Q_{ij} \equiv Y^2 \pmod{N}$$

abschließen zu können, in der Hoffnung, dass die algebraische Faktorisierung der Kongruenz etwas Nichttriviales ergibt.

12.2.1 Der Algorithmus

Um N zu faktorisieren, berechnen wir zuerst $R = \lfloor \sqrt{N} \rfloor$.

Wir setzen dann einen langen Vektor L für Subskripte n auf, der $-k$ bis k läuft, dessen Werte die *Logarithmen* von

$$Q_n = (R + n)^2 - N$$

sind und wir stellen fest, dass wir, genau wie bei CFRAC, die Werte haben, die modulo N quadrieren, um die Q_n zu testen, die wir auf Glätte testen werden. Wie bei CFRAC, wenn wir eine Menge S der Q_n sammeln können, deren Produkt eine perfekte Quadratzahl ist, dann haben wir die gewünschte Kongruenz

$$Y^2 = \prod_{n \in S} Q_n \equiv \prod_{n \in S} (R + n)^2 = X^2$$

Wir wählen, wie zuvor, eine Faktorbasis FB aus kleinen Primzahlen p .

Nun bestimmen wir für jedes $p \in FB$ einen Einstiegspunkt n_0 für den

$$(R + n_0)^2 - N = Q_{n_0} \equiv 0 \pmod{p}$$

und für alle Subskripte $n_0 \pm kp$ subtrahieren wir $\log p$ vom Array L .

Wenn wir mit der Faktorbasis fertig sind und Einträge in L haben, die null oder nahe null sind (wir müssen Rundungen von Gleitkommawerten zulassen), dann entsprechen diese wahrscheinlich Q_n , die vollständig über die Faktorbasis faktorisieren. Wir faktorisieren diese Werte erneut mit Hilfe der Probedivision.

Wir führen dann den Schritt der linearen Algebra durch, genau wie wir es für CFRAC getan haben.

12.2.2 Die entscheidenden Gründe für Erfolg und Verbesserung gegenüber CFRAC

Die Werte Q_n sind klein (genau wie bei CFRAC), weil wir $R + n$ fast gleich \sqrt{N} haben, so dass die Q_n nicht wirklich viel größer sind als N^2 .

Was jedoch entscheidend zu beachten ist und was den QS erfolgreich macht, ist, dass wir den Werten, die wir auf Glätte testen, eine Struktur gegeben haben. Sie liegen in einem Array, und wir können *mit konstantem Schritt durch dieses Array sieben*. Nicht alle Werte werden sich als glatt erweisen, aber wir haben teure Ganzzahlproben, die normalerweise fehlschlagen, durch Subtraktion mit konstantem Schritt ersetzt, nur an den Stellen, an denen eine Subtraktion notwendig ist, während wir durch ein sehr langes Array gehen. Da wir wissen, welche Array-Positionen eine $\log p$ subtrahiert haben müssen, ist kein Rechenschritt völlig verschwendet. Die einzige „verschwendete“ Berechnung ist die Subtraktion an Stellen, die sich letztendlich nicht als glatte Ganzzahlen herausstellen.

Wir können die Dinge beschleunigen (vielleicht), indem wir alle Fließkommawerte skalieren und dann feste Punkt-Ganzzahlarithmetik verwenden.

Wir stellen fest, dass wir tatsächlich die *Normen* algebraischer Ganzzahlen auf Glätte testen

$$\frac{-(R + n) + \sqrt{N}}{2}$$

im quadratischen Zahlbereich $\mathbb{Q}(\sqrt{N})$ (oder vielleicht $4N$; wir werden nicht pedantisch über Diskriminante N oder $4N$ sein). Dieses Thema wird fortgesetzt; das multiple Polynom-Quadrat-Sieb verwendet eine Anzahl von quadratischen Zahlbereichen mit N als Faktor der Diskriminante, und das allgemeinere Zahlbereichssieb, jetzt die beste der Faktorisierungsmethoden, tut dasselbe mit Normen in Bereichen höheren Grades als 2.

12.3 Noch einmal in die Bresche

Also lassen Sie uns 1037 noch einmal faktorisieren ...

$$R = \lfloor \sqrt{1037} \rfloor = 32$$

n	$R + n$	Q_n	Faktorisierung
-8	24	-461	$-1 \cdot 461$
-7	25	-412	$-1 \cdot 2^2 \cdot 10311$

-6	26	-361	$-1 \cdot 2^2 \cdot 19 \cdot 19$
-5	27	-308	$-1 \cdot 2^2 \cdot 7 \cdot 11$
-4	28	-253	$-1 \cdot 11 \cdot 23$
-3	29	-196	$-1 \cdot 2^2 \cdot 7^2$
-2	30	-137	$-1 \cdot 137$
-1	31	-76	$-1 \cdot 2^2 \cdot 17$
0	32	-13	$-1 \cdot 13$
1	33	52	$2^2 \cdot 13$
2	34	119	$7 \cdot 17$
3	35	188	$4 \cdot 47$
4	36	259	$7 \cdot 37$
5	37	332	$4 \cdot 83$
6	38	407	$11 \cdot 37$
7	39	484	$4 \cdot 121$
8	40	563	563
9	41	644	$4 \cdot 7 \cdot 23$

Wir sehen, dass $n = -3, 0, 1$ produziert

$$29^2 \cdot 32^2 \cdot 33^2 \equiv 364^2 \pmod{1037}$$

und dass $30624 - 364 = 30260$ den Faktor 17 und $30624 + 364 = 30988$ den Faktor 61 hat.

12.4 Das Mehrfachpolynom Quadratische Sieb

Die entscheidende Verbesserung bei CFRAC war die Vorstellung von Faktorbasis und der linearen Algebra, um eine $X^2 \equiv Y^2 \pmod{N}$ Kongruenz zu erhalten.

Die entscheidende Verbesserung bei QS bestand darin, die Glättungstests in eine Vektorsieboperation umzuwandeln.

Das Problem bei QS ist, dass wir nur ein quadratisches Polynom $X^2 - N$ haben, dessen Werte wir klein halten wollen, um sie wahrscheinlicher über die Faktorbasis glatt zu machen. Da dies eine Parabel ist, die nach rechts und links von $X_0 = \sqrt{N}$ ansteigt, werden die Werte stetig zunehmen, je weiter wir uns von der Quadratwurzel von N entfernen, und die Ausbeute an glatten Werten wird abnehmen.

Lassen Sie uns dieses Problem beheben.

Das zu faktorisierende N ist groß, vielleicht 1024 Bits (300 Dezimalstellen) oder mehr. Das Array, mit dem wir arbeiten, ist durch den physischen Speicher in einem

Computer begrenzt, so dass wir für eine Maschine mit 4 Gigabyte Platz für vielleicht 10^9 Dinge haben, was im Vergleich winzig ist.

Wir gehen nicht auf alle Details der Mathematik ein, aber das Wesentliche des Ansatzes ist folgendes. Anstatt mit dem einen Polynom $X^2 - N$ zufrieden zu sein, können wir ein wenig Calculus anwenden, um die Auswahl der Polynome $aX^2 + bXY + cY^2$ von Diskriminanten $b^2 - 4ac = kN$ für kleine Werte von k zu optimieren, so dass wir, wenn wir das Quadrat vervollständigen,

$$(2an + b)^2 - kN$$

klein für einen großen Bereich von Werten n haben.

Die Details sind in Silvermans ursprünglichem Papier [1] sehr klar. Im Wesentlichen nutzt man die Tatsache, dass es viele Polynome gibt und somit viele Polynome mit langen Reihen von kleinen Normwerten zum Testen auf Glätte. Wenn ein Polynom zu große Werte annimmt, wechselt das Programm zu einem anderen Polynom und fährt wie zuvor fort. Das Ziel ist schließlich, so viele glatte Zahlen wie möglich für die spätere Verwendung im linearen Algebra-Schritt zu ernten.

Alles andere verläuft wie zuvor. Wir laufen Gefahr, für kryptographische Zahlen $N = pq$, dass wir eine triviale Faktorisierung $k \cdot N$ anstelle von $(kp) \cdot (q)$ (oder vielleicht sogar $(k_1) \cdot (k_2N)$ für eine Faktorisierung von $k = k_1k_2$) erhalten, aber das ist ein Risiko, das wir eingehen, und wenn wir ausreichend viele glatte Q_n haben, werden wir schließlich eine Faktorisierung erhalten, die das p vom q trennt.

Die MPQS wurde von Lenstra und Manasse [2] bei der ersten Faktorisierung einer „harten“ Zahl von 100 Dezimalstellen verwendet.

12.4.1 Noch ein weiterer Vorteil

Ein letzter Vorteil, den die MPQS bietet und der von Silverman ausgiebig genutzt wurde und das Prototyp aller peinlich parallelen Crowdsourcing von Rechenkapazität (GIMPS, SETI At Home usw.) war, ist, dass dies eine peinlich parallele Berechnung ist, die auf Computern durchgeführt werden kann, die nur lose in einem Netzwerk gekoppelt sind. Da N groß ist, können wir davon ausgehen, dass alle Polynome zu einem anderen Bereich von Zahlen führen, die auf Glätte getestet werden, so dass es keine Überschneidungen oder Redundanzen bei der parallelen Glättungstestung gibt. Selbst wenn es eine Redundanz gäbe, würde dies die Fähigkeit der Berechnung, schließlich eine Faktorisierung zu erhalten, nicht ändern, sondern nur ihren Fortschritt verlangsamen. Silvermans Erfahrung war, dass das häufige Wechseln der Polynome sehr vorteilhaft war, so dass er die verschiedenen Berechnungen auf eine Reihe von verschiedenen Maschinen auslagerte. Da relativ wenige Reste glatt auftauchen, sind die Daten der glatten Zahlen, die an eine zentrale Stelle zurückkommen, klein im Vergleich zur Siebearbeit, die auf den

verschiedenen Computern geleistet wird, so dass wir keine Verbindung mit hoher Bandbreite zu all den verschiedenen Computern benötigen. Diese Nutzung von peinlicher Parallelität ist für solche Berechnungen zur Routine geworden.

12.5 Die Zahlkörpersieb

Das MPQS wurde Anfang der 1990er Jahre durch das Zahlkörper Sieb [3], auf das wir nicht im Detail eingehen werden, weil die Algebra viel komplizierter ist, abgelöst.

Es genügt zu sagen, dass der grundlegende Ansatz zur Faktorisierung im NFS genauso ist wie im MPQS und QS. Wir sieben, um Zahlen auf Glätte über einer Faktorbasis zu testen, indem wir Formeln verwenden, die uns eine Seite einer Kongruenz

$$X^2 \equiv Y^2 \pmod{N}$$

kostenlos geben und dann führen wir lineare Algebra modulo 2 durch, um eine Teilmenge unserer glatten Zahlen zu finden, die sich zu einer perfekten Quadratzahl multiplizieren. Schließlich testen wir

$$\gcd(X - Y, N)$$

und

$$\gcd(X + Y, N)$$

in der Hoffnung, einen Faktor zu finden.

Die Verbesserung, die das NFS gegenüber dem MPQS bietet, ist die gleiche wie die des MPQS gegenüber dem QS. Wir nutzen mehr von der algebraischen Struktur der Zahlkörper, um eine bessere Menge kleiner Zahlen zum Testen auf Glätte zu erhalten, so dass wir schneller mehr solcher glatten Zahlen bekommen.

Asymptotisch läuft das Allgemeine Zahlkörpersieb in der Zeit

$$L[1/3, (64/9)^{1/3}] = \exp\left(\left((64/9)^{1/3} + o(1)\right)(\ln N)^{1/3}(\ln \ln N)^{2/3}\right)$$

Zeit, wenn es zur Faktorisierung einer ganzen Zahl N verwendet wird.

12.6 Übungen

1. (Programmierübung.) Codieren Sie den Siebteil des quadratischen Siebs und verwenden Sie ihn, um zunächst 1037 wie im Beispiel, dann 1537 und 43039 zu faktorisieren. Bei 1037 und 1537 muss man nicht viel über die Faktorbasis nachdenken, aber bei 43039 und 1000009 muss man mit der Größe der Faktorbasis und der Länge des Siebarrays herumspielen.

2. (Programmierübung.) Nachdem Sie das quadratische Sieb codiert und es verwendet haben, um $N = 1000009$ zu faktorisieren, versuchen Sie, kN für kleine k zu sieben und zu sehen, ob Sie die gleiche Anzahl glatter Zahlen sammeln können, aber mit einem kürzeren Siebarray.

Literatur

1. R.D. Silverman, The multiple polynomial quadratic sieve. Math. Comput. **48**, 329–339 (1987)
2. A.K. Lenstra, M.S. Manasse, Factoring by electronic mail, in *Advances in cryptology – Eurocrypt '89*, Hrsg. von J.-J. Quisquater, J. Vandewalle (1990), S. 355–371
3. A.K. Lenstra, H.W. Lenstra Jr., *The development of the number field sieve*, Bd. 1554, Lecture notes in mathematics (Springer, Berlin, 1993)



Zyklen, Zufälligkeit, diskrete Logarithmen und Schlüsselaustausch

13

Zusammenfassung

Bei symmetrischer Kryptographie ist es notwendig, dass die beiden Parteien, die kommunizieren möchten, Zugang zu einem gemeinsamen Schlüssel haben, damit eine Partei eine Nachricht verschlüsseln und die andere Partei die Nachricht entschlüsseln kann. Dies würde die Fähigkeit von zwei Parteien, die in der Vergangenheit nicht kommuniziert haben, einschränken, die Art von sicherer Kommunikation zu führen, die beispielsweise für den elektronischen Handel notwendig ist. In diesem Kapitel beschreiben wir, wie zahlentheoretische Konstrukte, die scheinbar zufällige Zahlenfolgen erzeugen, verwendet werden können, um zwei Parteien den Austausch von Informationen zu ermöglichen, die es ihnen erlauben würden, sich auf einen gemeinsamen kryptographischen Schlüssel zu einigen, selbst wenn zwischen ihnen keine andere Kommunikation stattgefunden hat. Dieser Austausch von Schlüsselinformationen kann durch Exponentiation modulo einer großen Primzahl erfolgen, auf eine Weise ähnlich der RSA-Verschlüsselung, oder unter Verwendung von elliptischen Kurvengruppen in gleicher Weise. Wir werden auch die Grundlagen der Indexkalkulationsmethode behandeln, die, wenn auch mit Schwierigkeiten, zur Angriff auf diese Art des Schlüsselaustauschs verwendet werden kann.

13.1 Einführung

Das klassische Problem in der Kryptographie war schon immer der Wunsch von zwei Parteien, miteinander zu kommunizieren, aber anderen das Lesen und Verstehen dieser Kommunikation zu verhindern. Mit der Ausweitung der Informatik und der Entwicklung von Computernetzwerken ist eine neue Version dieses Problems von entscheidender Bedeutung geworden: Wie können zwei Parteien (sagen wir Armadillo und Bobcat), die

sich nie getroffen haben und deren einzige Verbindung eine erstmalige Kommunikation über Computer im Internet ist, ihre Identitäten gegenseitig authentifizieren? Wie kann eine sichere und asymmetrische Kommunikationsverbindung hergestellt werden, damit sie Informationen austauschen können, die von anderen Entitäten nicht gelesen werden können?

Wir bemerken, dass dies ein spezifisches Problem für asymmetrische Verschlüsselungsalgorithmen ist. In einer symmetrischen Welt hätten sowohl Armadillo als auch Bobcat einen kryptographischen Schlüssel geteilt und die Fähigkeit eines jeden, dem anderen eine bedeutungsvolle Kommunikation zu liefern, hätte gezeigt, dass die Partei am Ende der Kommunikationsverbindung Zugang zum Schlüssel hatte. (Wir geben zu, dass Abfangen, Nötigung und dergleichen einen solchen legitimen Gebrauch des Schlüssels kooptieren könnten, aber in der Tat besteht diese Verwundbarkeit bei jedem Versuch einer sicheren Kommunikation zwischen den Parteien.)

Und wir stellen von vornherein fest, dass dies ein grundlegendes Problem im elektronischen Handel ist. Wir kaufen Dinge online, aber wir tätigen diese Einkäufe manchmal bei Anbietern, mit denen wir zuvor noch nie Geschäfte gemacht haben.

Die derzeitige Lösung für dieses Kommunikationsproblem scheint immer wieder auf die Verwendung eines diskreten Logarithmus zurückzuführen zu sein. In diesem Kapitel präsentieren wir die grundlegende Mathematik hinter diskreten Logarithmen in mehreren relevanten Gruppen.

Wir bemerken, dass es, genau wie bei der Faktorisierung, eine umfangreiche Literatur über die Berechnung von diskreten Logs gibt [1–7].

13.2 Das diskrete Logarithmusproblem

Definition 13.1 Lassen Sie G eine zyklische Gruppe mit einem bekannten Generator g sein. Wenn die Gruppe zyklisch ist, dann kann jedes Element $a \in G$ als eine bestimmte Potenz g^e in G geschrieben werden. Gegeben ein solches Element $a \in G$, ist das *diskrete Logarithmusproblem* in G die Bestimmung des ganzzahligen Exponenten e , so dass $g^e = a$.

Die Verwendung des Wortes „diskret“ bezieht sich auf den gewöhnlicheren Logarithmus, bei dem wir schreiben würden

$$\begin{aligned} g^e &= a \\ e &= \log_g a \end{aligned}$$

Einige diskrete Logarithmus (DL) Probleme sind einfach. Zum Beispiel, lassen Sie die Gruppe G die Gruppe der ganzen Zahlen modulo n unter Addition sein. Der Generator ist $g = 1_G$. Gegeben jedes Element m_G in der Gruppe, ist es trivial zu sehen, dass

$$m_G = m \cdot 1_G$$

wobei wir 1_G und m_G indiziert haben, um diese als Elemente in der Gruppe zu identifizieren, die sich von dem m auf der rechten Seite unterscheiden, das die wiederholte Anwendung m mal der Gruppenoperation darstellt.

Wir können auch unsere Einschränkungen etwas lockern. Wir werden Gruppen betrachten, für die die gesamte Gruppe zyklisch ist, so dass die Potenzen des Generators die gesamte Gruppe erzeugen. Was wir wirklich brauchen, ist nur, dass der Zyklus, der durch die Potenzen von g erzeugt wird, ausreichend groß ist, dass das diskrete Logarithmusproblem schwer ist.

13.3 Schwierige Diskrete Logarithmusprobleme

Es ist eine *äußerst wichtige Tatsache*, dass das diskrete Logarithmusproblem für eine Reihe nützlicher Gruppen schwer zu lösen ist, weil das Potenzieren eines Generators g die Gruppe in einer ziemlich zufälligen Reihenfolge durchläuft. Für die Ganzzahlen unter Addition ist das DL-Problem trivial. Für die Ganzzahlen modulo einer Primzahl unter Multiplikation ist das Problem jedoch sehr schwer.

Was wir sehen werden, ist, dass das diskrete Logarithmusproblem ein schwieriges Problem (nach irgendeiner Definition von „schwer“) in vielen Gruppen ist, für die die Exponentiation/Rechnung einfach ist. Das heißt, die Exponentiation zu machen, um $a = g^e$ gegeben g und e ist einfach, aber die Exponentiation rückgängig zu machen, um e aus a und g zu bekommen, ist schwer.

Zum Beispiel, modulo 11, mit primitiver Wurzel 2, haben wir

$$2^1 \equiv 2$$

$$2^2 \equiv 4$$

$$2^3 \equiv 8$$

$$2^4 \equiv 5$$

$$2^5 \equiv 10$$

$$2^6 \equiv 9$$

$$2^7 \equiv 7$$

$$2^8 \equiv 3$$

$$2^9 \equiv 6$$

$$2^{10} \equiv 1$$

In diesem Fall, weil 2 klein ist, können wir einige der Sequenzen vorhersagen (2 bis 4 bis 8, zum Beispiel). Aber wenn wir eine primitive Wurzel wählen, für die das „Umschlagen“ modulo 11 fast garantiert ist, sagen wir 7, dann gibt es (anscheinend) keine einfache Möglichkeit, der Sequenz zu folgen:

$$\begin{aligned}
7^1 &\equiv 7 \\
7^2 &\equiv 5 \\
7^3 &\equiv 2 \\
7^4 &\equiv 3 \\
7^5 &\equiv 10 \\
7^6 &\equiv 4 \\
7^7 &\equiv 6 \\
7^8 &\equiv 9 \\
7^9 &\equiv 8 \\
7^{10} &\equiv 1
\end{aligned}$$

Dies ist die Schlüsseltatsache, die in vielen kryptographischen Schemata verwendet wird.

13.4 Zyklen

Wir haben in Abschn. 4.4.1 das Konzept einer primitiven Wurzel erwähnt. Lassen Sie uns damit fortfahren.

Definition 13.2 Eine *primitive Wurzel* in einer zyklischen Gruppe G der Ordnung n ist ein Element g , das der Ordnung n ist.

Satz 13.1 Für n eine Ganzzahl ist die Ordnung der multiplikativen Gruppe der Ganzzahlen modulo n $\phi(n)$.

Satz 13.2 Für p eine Primzahl und k eine positive Ganzzahl haben wir $\phi(p^k) = (p-1)p^{k-1}$.

Satz 13.3 Die einzigen Ganzzahlen n , für die primitive Wurzeln existieren, sind 2, 4, Primzahlpotenzen p^k für k eine positive Ganzzahl, und $2p^k$ für k eine positive Ganzzahl.

Bemerkung 13.1 Wir kehren zurück zur multiplikativen Struktur der Ganzzahlen mod 15 für ein Beispiel dieser Ergebnisse. Da $15 = 3 \cdot 5$, haben wir Da $\phi(15) = \phi(3)\phi(5) = 2 \cdot 4 = 8$. Da 15 nicht die im Satz 13.3 erwähnte Form hat, ist es nicht multiplikativ eine zyklische Gruppe, sondern stattdessen das Produkt eines 2-Zyklus (aus der 3) und eines 4-Zyklus (aus der 5). Die multiplikative Gruppe ist das direkte Produkt

$$\{1, 11\} \times \{1, 2, 4, 8\}$$

Satz 13.4 Es gibt genau $\phi(p-1)$ primitive Wurzeln der multiplikativen Gruppe der Ganzzahlen modulo einer Primzahl p .

Beweis Um dies zu sehen, betrachten wir die Potenzen der primitiven Wurzel in Exponentenordnung

$$g^1, g^2, g^3, \dots, g^{p-1} = 1$$

und wir erinnern uns daran, dass das Multiplizieren dieser Elemente dasselbe ist wie das Addieren der Exponenten.

Offensichtlich wird ein Exponent e , der prim zu $p-1$ ist, additiv einen vollständigen Zyklus der Exponenten 1 bis $p-1$ erzeugen, und ein Exponent e , der Faktoren gemeinsam mit $p-1$ hat, wird einen Kurzzyklus erzeugen. \square

13.5 Cocks-Ellis-Williamson/Diffie-Hellman Schlüsselaustausch

Nehmen wir an, dass Armadillo und Bobcat geheime Informationen über einen unsicheren Kommunikationskanal austauschen möchten. Dies wäre einfach für sie zu tun, wenn sie einen kryptographischen Schlüssel zur Verschlüsselung hätten. Aber was, wenn sie sich noch nie getroffen haben? Was, wenn dies ihre erste Interaktion ist? Was, wenn Armadillo ein normaler Benutzer ist und Bobcat ein großes kommerzielles Unternehmen, bei dem Armadillo Waren kaufen möchte?

Die Grundlage für die Erstellung eines gemeinsamen Schlüssels, den sowohl Armadillo als auch Bobcat haben, der aber rechnerisch unpraktikabel für andere zu bestimmen ist, ist das Schlüsselaustauschprotokoll, das von Cocks, Ellis und Williamson erfunden wurde (wenn man britisch ist) oder von Diffie und Hellman (wenn man aus den Vereinigten Staaten kommt).

13.5.1 Der Schlüsselaustausch-Algorithmus

Armadillo bestimmt eine Primzahl P , eine primitive Wurzel g für die Gruppe der Ganzzahlen mod P und einen geheimen Exponenten e_A . Armadillo veröffentlicht die Primzahl P und die primitive Wurzel g als öffentliche Informationen, berechnet $m_A \equiv g^{e_A} \pmod{P}$ und sendet m_A an Bobcat.

Bobcat liest die Werte von P , g und m_A und bestimmt ihren eigenen geheimen Exponenten e_B . Sie berechnet $m_B \equiv g^{e_B} \pmod{P}$ und sendet das an Armadillo.

Armadillo verwendet Bobcats übermitteltes m_B um zu berechnen, unter Verwendung ihres Geheimnisses e_A ,

$$S = (m_B)^{e_A} \equiv (g^{e_B})^{e_A} \equiv g^{e_A e_B} \pmod{P}.$$

Bobcat ihrerseits verwendet Armadillos übermitteltes m_A um zu berechnen, unter Verwendung ihres Geheimnisses e_B ,

$$S = (m_A)^{e_B} \equiv (g^{e_A})^{e_B} \equiv g^{e_A e_B} \pmod{P};$$

und Armadillo und Bobcat haben nun ein gemeinsames Geheimnis S .

Ein Außenstehender, der nur P , g , m_A und m_B gesehen hat, kann die Berechnung von Armadillo oder Bobcat nicht reproduzieren, ohne das diskrete Logarithmusproblem modulo P zu lösen, um einen oder den anderen der geheimen Exponenten e_A oder e_B zu bestimmen. (Oder zumindest ist öffentlich keine Methode bekannt, um e_A oder e_B ohne Lösung des diskreten Logarithmusproblems zu finden.)

Das CEW/DH-Protokoll wird oft verwendet, um einen Schlüssel in einer kryptographischen Umgebung auszutauschen, daher der Begriff „Schlüsselaustausch“. Tatsächlich wird ein großer Teil des elektronischen Handels genau auf diese Weise abgewickelt; das diskrete Logarithmusproblem wird verwendet, damit zwei Parteien sich auf einen geheimen Schlüssel einigen können, und dann wird der Schlüssel in AES verwendet, weil AES als sicher gilt und schnell ist.

Wir bemerken gute Wahlmöglichkeiten für Primzahlen P . Eine *sichere Primzahl* ist eine Primzahl P , für die $(P - 1)/2$ ebenfalls eine Primzahl ist. Sichere Primzahlen haben den längstmöglichen Zyklus im Vergleich zur Anzahl der Bits in der Primzahl.

13.6 Der Index-Kalkül

Die Lösung eines diskreten Logarithmusproblems modulo einer Primzahl P verwendet den *Index-Kalkül* und steht in Zusammenhang mit den Siebmethoden zur Faktorisierung. Die Idee geht wieder auf Kraitchik zurück, wurde aber in den späten 1970er Jahren neu erfunden, als kryptographische Anwendungen die Berechnungsanzahlentheorie plötzlich modischer machten.

Die klassische Beschreibung des Algorithmus stammt von Coppersmith, Odlyzko und Schroepel [8], üblicherweise als C-O-S bezeichnet; einige zusätzliche Referenzen stammen von LaMacchia [9] und Odlyzko [10]. Eine etwas intuitive Vorstellung von dem Algorithmus ist diese:

- die Ganzzahlen haben eine multiplikative Basis der Primzahlen, insofern als jede Ganzzahl ein Produkt von Primzahlen ist;
- wenn wir die entsprechenden Logarithmen mod P von ausreichend vielen kleinen Primzahlen bestimmen können, dann können wir die Logarithmen in der Faktorisierung einer Ganzzahl M addieren, um den diskreten Logarithmus modulo P zu erhalten.

Wir veranschaulichen den Algorithmus mit einem Beispiel, unter Verwendung der C-O-S-Beschreibung. Obwohl es rechnerisch bessere Methoden gibt, erfordern diese mehr

mathematischen Hintergrund, verbessern aber nicht wirklich das Verständnis dafür, wie der Algorithmus funktioniert.

13.6.1 Unser Beispiel

Wir wählen 1019 als unsere Primzahl und stellen fest, dass $(1019 - 1)/2 = 509$ ebenfalls eine Primzahl ist, so dass der Zyklus der Potenzen so lang wie möglich wäre. Wir stellen fest, dass 2 eine primitive Wurzel ist.

13.6.2 Glatte Beziehungen

Wir berechnen $H = \lfloor \sqrt{1019} \rfloor + 1 = 32$. Wie bei Siebmethoden zur Faktorisierung wählen wir eine Faktorbasis $Q = \{q_i\}$ aus kleinen Primzahlen $-1, 2, 3, 5, 7, 11, 13$ (einschließlich -1 als „Primzahl“, genau wie bei den Faktorisierungsalgorithmen).

Wir führen nun eine Doppelschleife auf c_1 und c_2 für kleine Werte von c_1, c_2 durch, berechnen

$$(H + c_1)(H + c_2) \pmod{P}$$

und versuchen, diese (die Analogien der Q_i im Sieb zur Faktorisierung) über die Faktorbasis zu faktorisieren. Da H etwa \sqrt{P} beträgt, wird dieses Produkt etwa P groß sein, so dass diese Produkte im Vergleich zu P selbst „klein“ sein sollten.

Für die Produkte, die sich faktorisieren lassen, haben wir

$$\prod q_i^{e_i} \equiv (H + c_1)(H + c_2) \pmod{P}$$

und somit

$$\sum_i e_i \log q_i - \log(H + c_1) - \log(H + c_2) \equiv 0 \pmod{P - 1}$$

Wir erweitern unsere Faktorbasis um die $H + c_1$ und $H + c_2$, und wir fügen das eine inhomogene Logarithmus hinzu, von dem wir sicher sind:

$$\log 2 = 1,$$

weil wir 2 als unsere primitive Wurzel verwenden.

13.6.3 Matrix-Reduktion

Wenn wir die Doppelschleife auf c_1 von -5 bis 4 und auf c_2 von $c_1 + 1$ bis 5 , einschließlich, ausführen und die Faktorisierungen durchführen, erhalten wir das folgende Tableau, in dem

wir die Spalten der Faktorbasis und die Spalten von der $H + c_i$ zur besseren Lesbarkeit getrennt haben.

-1	2	3	5	7	11	13	27	28	29	30	31	32	33	34	35	36	37	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	1	0	1	-1	0	0	0	-1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	-1	0	0	0	1	-1	0	0	0	0	0	0
1	7	0	0	0	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0
1	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	-1	0	1	0
1	2	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	-1	0
1	0	1	0	0	0	1	0	-1	0	0	0	0	0	0	-1	0	0	0
1	0	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	-1	0	0
1	3	1	1	0	0	0	0	0	-1	0	-1	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	-1	0	0	-1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	-1	0	1	0	-1	0	0	0	0	0
1	0	1	0	0	1	0	0	0	-1	0	0	0	0	-1	0	0	0	0
1	2	0	0	0	0	0	0	0	-1	0	0	0	0	0	-1	0	0	0
0	0	0	2	0	0	0	0	0	-1	0	0	0	0	0	0	-1	0	0
0	1	3	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	-1	0
1	0	0	0	0	0	0	0	0	1	-1	0	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	0	-1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	-1	0	0	0
0	0	0	0	1	0	1	0	0	0	-1	0	0	0	0	0	0	-1	0
1	0	3	0	0	0	0	0	0	0	0	-1	-1	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0	0	-1	0	-1	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	-1	0	0	-1	0	0	0	0
0	1	1	0	0	1	0	0	0	0	0	-1	0	0	0	-1	0	0	0
0	7	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	-1	0
0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	0	0	0	1	0
0	0	1	1	0	1	0	0	0	0	0	0	-1	0	0	0	0	-1	0
0	0	0	0	0	0	2	0	0	0	0	0	0	-1	0	0	-1	0	0

Die erste Zeile unter der Überschrift ist die inhomogene $\log 2 = 1$. Die zweite Zeile stellt das Faktorisieren dar

$$-1 \cdot 2 \cdot 7 \cdot 13 = -182 \equiv 27 \cdot 31 = (32 - 5)(32 - 1) \pmod{1019}$$

Wenn wir wollen, können wir beim Durchführen eines Beispiels schummeln und die Logarithmen aus der primitiven Wurzel berechnen:

-1	2	3	5	7	11	13	27	28	29	30	31	32	33	34	35	36	37	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	701
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Dies sind die Werte, die wir aus unserem kleinen Beispiel-Spickzettel erhalten haben, daher wissen wir, dass wir diese Matrixreduktion korrekt durchgeführt haben. Unsere Reduktion war fast nur eine Gaußsche Elimination. Wir schreiben „fast“, weil wir eine Reduktion modulo einer zusammengesetzten Ganzzahl durchführen und nicht garantieren können, dass wir teilen können, daher wenden wir stattdessen im Wesentlichen einen ggT-Prozess in der Zeilenreduktion an. Bei einem echten Problem würden ausgefeiltere Matrixreduktionstechniken, normalerweise eine Variante der Lanczos-Methode, verwendet werden.

13.6.4 Einzelne Logarithmen

Nachdem wir die Logarithmen für die Elemente unserer (erweiterten) Faktorbasis erstellt haben, besteht das letztendliche Problem darin, einen einzelnen Logarithmus zu berechnen. Wir nehmen an, dass wir einen Wert b haben und dass wir den Logarithmus x berechnen möchten, so dass $g^x \equiv b \pmod{P}$ in unserem Beispiel, wo wir $g = 2$ als unsere primitive Wurzel gewählt haben. Zu diesem Zweck berechnen wir

$$g^wb \pmod{P}$$

für einen zufällig gewählten Wert von w , und wir zerlegen dies in

$$g^wb \equiv \prod q_i^{e_i} u_i^{f_i} \pmod{P}$$

für einige „mittelgroße“ Primzahlen u_i . Dies gibt uns mehr Werte, für die wir die Logarithmen finden können, genau wie wir die Faktorbasis der kleinen Primzahlen erweitert haben, um die $H + c_j$ einzuschließen. Wir können das oben genannte Sieben und die Matrixreduktion wiederholen, um die Logarithmen der u_i zu finden, und dann wird der Logarithmus von b bestimmt. Wir haben dies mit unserem Beispiel nicht

gemacht, weil die Zahlen so klein sind, dass „mittelgroß“ wenig Aufschluss geben würde.

13.6.5 Asymptotik

Die einfache Version eines Index-Kalkül-Algorithmus läuft in der Zeit

$$L[1/2, c] = \exp((c + o(1))(\ln N)^{1/2}(\ln \ln N)^{1/2})$$

Zeit, für eine konstante c , aber dies kann verbessert werden, indem man ein Analogon des Number Field Sieve zur gleichen $L[1/3, c]$ Laufzeit (mit einer anderen konstanten c) wie das NFS selbst verwendet.

13.7 Schlüsselaustausch mit elliptischen Kurven

Beachten Sie, dass der oben beschriebene Schlüsselaustausch nicht wirklich ein Ergebnis ist, das nur von den ganzen Zahlen modulo P abhängt. Es handelt sich tatsächlich um ein Ergebnis über Berechnungen in zyklischen Gruppen, für die das diskrete Logarithmusproblem schwierig ist. Wie bei RSA ist es im Vergleich zu anderen Gruppen, die für die Verschlüsselung verwendet werden könnten, einfach so, dass die Verwendung der ganzen Zahlen mod P rechnerisch sehr einfach ist als Grundlage dafür, wie ein solcher Algorithmus funktionieren sollte. Bei der Abwägung der Komplexität von Berechnungen mod P gegen die Schwierigkeit eines mod- P diskreten Logarithmusproblems ist es sehr schwierig, eine andere Gruppe zu finden, deren diskretes Logarithmusproblem genauso schwierig ist, aber nicht kostspieliger für den Schlüsselaustausch ist. Andererseits gibt es den Indexkalkulus-Angriff, und dieser funktioniert nicht mit einigen der anderen Gruppen, die verwendet werden könnten (wie die Gruppe einer elliptischen Kurve).

Der Schlüsselaustausch mit elliptischen Kurven ist völlig analog zum grundlegenden Algorithmus mit einer großen Primzahl. Armadillo und Bobcat einigen sich auf eine elliptische Kurve \mathcal{E} modulo einer großen Primzahl P , so dass die Ordnung der Kurve zu schwierig zu berechnen ist, und einen Basispunkt Q auf dieser Kurve. Armadillo und Bobcat wählen ihre eigenen geheimen Exponenten e_A, e_B wie zuvor. Armadillo berechnet $e_A \cdot Q$ in der Kurvengruppe und sendet das an Bobcat. Bobcat berechnet $e_B \cdot Q$ in der Kurvengruppe und sendet das an Armadillo. Sie können nun beide $e_A \cdot e_B \cdot Q$ berechnen und den gleichen Punkt auf der Kurve erhalten, aus dessen x -Koordinate ein gemeinsamer geheimer Schlüssel abgeleitet werden kann.

Das herausragende Merkmal von elliptischen Kurvengruppen für den Schlüsselaustausch ist, dass es keinen offensichtlichen Indexkalkulus gibt. Die gewöhnlichen Ganzzahlen (nicht die Ganzzahlen modulo P) werden mit den Primzahlen als multiplikative Basis und dem Hauptsatz der Arithmetik erzeugt. Der Indexkalkulus funktioniert modulo

P , weil die multiplikative Erzeugung von Resten ausreichend Informationen mod P liefert, dass das Berechnen der Logarithmen von ausreichend vielen kleinen Primzahlen das Berechnen der Logarithmen für den Rest der Reste ermöglicht. Der zweite Schritt ist ein Bootstrap von einer eher kleinen Faktorbasis zu einer erweiterten Faktorbasis, die nur die Primzahlen enthält, die wir zufällig benötigen, nicht alle Primzahlen (das wären zu viele).

Diese Situation haben wir nicht bei elliptischen Kurven. Es gibt kein klares Analogon zum Indexkalkulus, weil es keine bekannte multiplikative Basis für die Punkte auf einer elliptischen Kurve modulo einer Primzahl gibt, obwohl Joseph Silverman einen Algorithmus vorschlug, den er als „Xedni-Kalkulus“ bezeichnete [11]. („Xedni“ ist „Index“ rückwärts.) Der Xedni-Kalkulus wurde nicht effektiv eingesetzt, obwohl er von einigem Interesse bleibt.

Was normalerweise für diskrete Logarithmenberechnungen in elliptischen Kurven verwendet wird, ist eine parallelisierbare Variante des Pollard-Rho-Algorithmus. Wir werden dies in Kap. 14 aufgreifen.

13.8 Schlüsselaustausch in anderen Gruppen

Genau wie bei Analogien zu RSA für die Kryptographie gibt es andere Gruppen (neben elliptischen Kurven), die Eigenschaften ähnlich denen der ganzen Zahlen modulo einer Primzahl haben, und diese Gruppen wurden für Schlüsselaustausch-Algorithmen vorgeschlagen. Die Klassengruppen von komplexen quadratischen Feldern sind im Allgemeinen fast zyklisch, und die Klassengruppe von Diskriminante N ist ungefähr \sqrt{N} groß. Genau wie bei den ganzen Zahlen modulo P ist die Exponentiation in Klassengruppen ein unkomplizierter Prozess, und die Reduktion von Formen (oder Idealen) zu reduzierten Formen/Idealen ähnelt dem „Wrap“ von Potenzen einer primitiven Wurzel modulo einer großen Primzahl. Allerdings kann ein Ansatz, der dem Indexkalkulus völlig ähnlich ist, für diese Gruppen verwendet werden, weil die Multiplikation von Klassen fast das Gleiche ist wie die Multiplikation der führenden Koeffizienten von Formen (es ist, wenn die Produkte, die die führenden Koeffizienten sind, reduziert werden, um die kanonische reduzierte Form der Klasse zu erzeugen, dass die Verschleierung des Produkts stattfindet).

Obwohl die kryptographische Sicherheit einer solchen mathematischen Struktur genauso gut sein könnte wie die der ganzen Zahlen modulo P , besteht das gleiche Problem für diese Gruppen wie für die Kryptographie selbst – die rechnerische Einfachheit der Multiplikation modulo einer Primzahl kann nicht mit den Kosten für das Zusammensetzen von Formen und Klassen verglichen werden, was mindestens eine ggT-Operation bei jeder Multiplikation erfordert. Das Niveau des zufälligen Verhaltens, das für die Kryptographie benötigt wird, ist vorhanden, aber die Kosten der Berechnungen sind viel größer als für die Arithmetik modulo großen Primzahlen, ohne wirklichen Nutzen für die Sicherheit.

Tab. 13.1 Rekordberechnungen diskreter Logarithmen modulo Primzahlen

Dezimalzahlen	Bits	Angekündigt	Notizen
130	431	18. Juni 2005	
160	530	5. Februar 2007	Sichere Primzahl
180	596	11. Juni 2014	Sichere Primzahl
232	765	16. Juni 2016	Sichere Primzahl
240	795	2. Dezember 2019	Sichere Primzahl

Tab. 13.2 Rekordberechnungen diskreter Logarithmen in elliptischen Kurven

Bits	Angekündigt
112	Juli 2009
113	April 2014
114	21. August 2017
114	Juni 2020

13.9 Wie schwierig ist das diskrete Logarithmusproblem?

Wie bei RSA lohnt es sich, einen Blick auf Rekordleistungen bei der Berechnung diskreter Logarithmen zu werfen. Die neuesten Rekorde für die Berechnung diskreter Logarithmen modulo Primzahlen sind in Tab. 13.1 dargestellt.

Für diskrete Logarithmen in elliptischen Kurvengruppen besteht eine Standardherausforderung, die von der Certicom Corporation ausgegeben wurde. Ihre Level I-Herausforderungen beinhalten Kurven modulo 109-Bit und 131-Bit Primzahlen, und die Level II-Herausforderungen haben Primzahlen von 163, 191, 239 und 359 Bits. Certicom behauptet, dass alle Level II-Herausforderungen derzeit nicht machbar sind. Rekorde für die Berechnung diskreter Logarithmen in elliptischen Kurven sind in Tab. 13.2 dargestellt.

Man kann sehen, im Vergleich der Rekordberechnungen, den inhärenten Vorteil elliptischer Kurven. Obwohl die Gruppenoperationen komplizierter sind als nur das Multiplizieren großer Zahlen, kann man vergleichbare Sicherheit erzielen, indem man elliptische Kurven mit viel kleineren Moduli als mit Primzahlenmoduli verwendet.

13.10 Übungen

1. (Mögliche Programmieraufgabe.) Richten Sie einen Schlüsselaustauschprozess ein, wie in Abschn. 13.5.1. Testen Sie zuerst mit der Primzahl 31, um sicherzustellen, dass Sie den Prozess verstehen. Versuchen Sie es dann mit der Primzahl 257.

2. (Programmieraufgabe.) Wenn Sie dies für die vorherige Aufgabe noch nicht getan haben, schreiben Sie den Code für den Schlüsselaustausch, wie in Abschn. 13.5.1. Testen Sie zuerst mit der Primzahl 31, um sicherzustellen, dass Sie den Prozess verstehen. Versuchen Sie es dann mit der Primzahl 257. Verwenden Sie schließlich die Primzahl 46301, die groß genug sein wird, um sicherzustellen, dass Sie den Code korrekt haben, aber keine mehrstellige Arithmetik erfordern wird, selbst wenn sie in C++ oder Java durchgeführt wird.
3. (Programmieraufgabe.) Schreiben Sie einfachen Code (das heißt, gehen Sie vor und verwenden Sie die Probedivision) für den Indexkalkül und überprüfen Sie das Beispiel von Abschn. 13.6.1.
4. (Programmieraufgabe.) Führen Sie den nächsten Schritt des Indexkalküls durch, indem Sie die Logarithmen von 41, 43 und 47 berechnen, unter Verwendung des in Abschn. 13.6.4 skizzierten Prozesses.

Literatur

1. T. Elgamal, A subexponential-time algorithm for computing discrete logarithms over $GF(p^2)$. IEEE Trans. Inf. Theory 473–481 (1985)
2. D.M. Gordon, Discrete logarithms in $GF(p)$ using the number field sieve. SIAM J. Discrete Math. **6**, 124–138 (1993)
3. D.M. Gordon, K.S. McCurley, Computation of discrete logarithms in fields of characteristic two. Crypto 91 rump session paper (1991)
4. J.M. Pollard, Monte Carlo methods for index computation mod p . Math. Comput. 918–924 (1978)
5. D. Weber, An implementation of the general number field sieve to compute discrete logarithms mod p . Proc. Eurocrypt **95**, 95–105 (1995)
6. D. Weber, Computing discrete logarithms with the general number field sieve. Proceedings, ANTS II (1996)
7. D. Weber, Computing discrete logarithms with quadratic number rings, in *Advances in cryptology – EUROCRYPT '98*, Bd. 1403, Lecture notes in computer science, Hrsg. von K. Nyberg (1998), S. 171–183
8. D. Coppersmith, A.M. Odlyzko, R. Schroepel, Discrete logarithms in $GF(p)$. Algorithmica 1–15 (1986)
9. B.A. LaMacchia, A.M. Odlyzko, Computation of discrete logarithms in prime fields. Des. Codes Cryptogr. **1**, 47–62 (1991)
10. A.M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, in *Advances in cryptology – EUROCRYPT '84*, Bd. 209, Lecture notes in computer science, Hrsg. von T. Beth, N. Cot, I. Ingemarsson (1985), S. 224–314
11. J.H. Silverman, The Xedni calculus and the elliptic curve discrete logarithm problem. Des. Codes Cryptogr. **20**, 5–40 (2000)



Zusammenfassung

Das erste vorgeschlagene asymmetrische Verschlüsselungsschema war das von Rivest, Shamir und Adleman, das die Exponentiation in der Gruppe der Ganzzahlen modulo das Produkt von zwei großen Primzahlen verwendet. Koblitz und Miller schlugen unabhängig voneinander die Verwendung der Gruppen von Punkten auf elliptischen Kurven vor. In diesem Kapitel behandeln wir den Algorithmus zur Verwendung von Kurven für die Kryptographie sowohl für die Verschlüsselung als auch für den Schlüsselaustausch. Da die Arithmetik zur Punktaddition aufwendig ist, enthalten wir die Formeln zur effizienten Addition von Punkten. Schließlich beinhalten wir den Pohlig-Hellman-Angriff, der bei richtig gewählten Kurven nicht erfolgreich sein sollte, und den Pollard-Rho-Angriff, der derzeit der beste Angriff auf das diskrete Logarithmusproblem der elliptischen Kurve ist.

14.1 Einführung

Die Verwendung von elliptischen Kurven in der Kryptographie wurde fast zur gleichen Zeit in den Mitte der 1980er Jahre unabhängig voneinander von Neal Koblitz [1] und Victor Miller [2] vorgeschlagen, und seit der Einführung dieser Idee hat es eine Explosion in der Untersuchung von Kurven gegeben.

Wir überprüfen die grundlegende (Schul-) Algebra, die eine elliptische Kurve definiert \mathcal{E} .

Wir beginnen mit einer polynomischen Gleichung, die quadratisch in Y und kubisch in X ist

$$\mathcal{E} : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

und, obwohl wir in Kürze eine Änderung des Grundfeldes vornehmen werden, betrachten wir dies als eine Gleichung mit rationalen Konstanten a_1, a_3, a_2, a_4, a_6 und rationalen Werten von X und Y . Die Kurve \mathcal{E} ist die Menge der Paare rationaler Zahlen (X, Y) zusammen mit dem *Punkt im Unendlichen* \mathcal{O} .

Der Schlüssel zum Verständnis von Kurven ist die Tatsache, dass wir, da wir rationale (X, Y) betrachten, rationale Variablenänderungen vornehmen dürfen, ohne die Menge der Punkte, die auf der Kurve liegen, zu ändern. Wenn wir die Variablenänderung vornehmen

$$Y' = Y + a_1/2X + a_3/2$$

erhalten wir

$$\mathcal{E} : (Y')^2 = X^3 + a_2'X^2 + a_4'X + a_6'$$

und dann mit

$$X' = X + a_2/3$$

erhalten wir

$$\mathcal{E} : (Y')^2 = (X')^3 + a_4''X' + a_6''.$$

Da diese Transformationen nicht beeinflussen, ob Lösungen rationale Zahlen sind oder nicht, können wir diese letzte

$$\mathcal{E} : Y^2 = X^3 + a_4X + a_6$$

als die kanonische Definition für eine elliptische Kurve über den Rationalzahlen nehmen.

VORSICHT: Wir werden Gelegenheit haben, Kurven über Feldern zu betrachten, die nicht die rationalen Zahlen sind, nämlich Felder modulo Primzahlen und endliche Felder der Charakteristik 2. Da wir große Primzahlen P verwenden werden, verursachen die Division durch 2 und durch 3 in diesen beiden Transformationen keine Probleme, aber über Felder der Charakteristik 2 kann die erste dieser Transformationen nicht durchgeführt werden, und wir werden eine kanonische Darstellung haben

$$\mathcal{E} : Y^2 + a_3Y = X^3 + a_4X + a_6$$

14.1.1 Jacobianische Koordinaten

Wir bemerken, dass obwohl die High-School-Algebra ein wenig sauberer erscheint, wenn wir die Kurven mit rationalen Zahlen schreiben, der Wechsel zu jacobianischen Koordinaten fast erforderlich ist, wenn wir uns tatsächliche Berechnungen ansehen. Die Kurve über den rationalen Zahlen, mit Lösungen x und y rational,

$$\mathcal{E} : y^2 = x^3 + a_4x + a_6,$$

ist wirklich die Kurve

$$\mathcal{E} : (Y/Z^3)^2 = (X/Z^2)^3 + a_4(X/Z^2) + a_6,$$

mit Lösungen X , Y und Z , die Ganzzahlen sind. Wir können den Bedarf an Nennern beseitigen, indem wir mit Z^6 multiplizieren, um

$$\mathcal{E} : Y^2 = X^3 + a_4XZ^4 + a_6Z^6,$$

in *jacobianischen Koordinaten* zu erhalten. Und jetzt sind alle X , Y und Z Ganzzahlen, und wir wissen, wie man mit Ganzzahlen rechnet.

14.2 Elliptische Kurve Diskrete Logarithmen

Die potenzielle Verwendung von elliptischen Kurven für diskrete Logarithmen und somit für den Schlüsselaustausch ist etwas Selbstverständliches, sobald man daran denkt, solche Dinge zu verwenden. Die Ordnung der Gruppe modulo einer großen Primzahl P ist bekanntlich „ungefähr“ P . Die Sequenz von Punkten Q , die durch Multiplikation eines bestimmten Basispunktes erzeugt wird, haben Koordinatenwerte, die „so zufällig“ erscheinen, wie es für kryptographische Zwecke benötigt würde. Und im Gegensatz zu den Gruppen modulo Primzahlen P unter Multiplikation, für die die Indexkalkulushmethode das diskrete Logarithmenproblem in subexponentieller Zeit löst, sind keine der Angriffe auf diskrete Logarithmen von elliptischen Kurven besser als \sqrt{P} , was bedeutet, dass viel kleinere P verwendet werden können.

Die Schlüsselfrage wäre, ob der zusätzliche Aufwand für die Addition und Verdoppelung der elliptischen Kurve es wert wäre. Wir werden uns die Kosten der Arithmetik in Kurven in Abschn. 14.4 ansehen, aber die Antwort im Allgemeinen ist ein eindeutiges „ja“.

14.3 Elliptische Kurven Kryptographie

Die Gruppe von Punkten auf einer elliptischen Kurve scheint eine vollkommen vernünftige Menge von „zufälligen“ Punkten zu sein, daher scheinen diese fruchtbare mathematische Konstrukte für den Schlüsselaustausch zu sein. Kryptographie mit elliptischen Kurven erfordert ein wenig mehr Erklärung.

Armadillo, der sicher kommunizieren möchte, wählt eine elliptische Kurve \mathcal{E} , einen Basispunkt P von großer Primordnung n , und macht diese öffentlich. Armadillo wählt auch einen privaten Schlüssel d und berechnet den öffentlichen Schlüssel

$$Q = dP.$$

Wenn Bobcat eine sichere Nachricht an Armadillo senden möchte, kennt sie \mathcal{E} , P , und Q , aber nicht d . Die Klartextnachricht m wird umgewandelt, um einen Punkt M auf der Kurve darzustellen. Bobcat berechnet ein zufälliges k und dann berechnet sie $R_1 = kP$ und $R_2 = M + kQ$ auf der Kurve und sendet diese an Armadillo.

Armadillo berechnet nun

$$dR_1 = dkP = kdP = kQ$$

und verwendet dies dann, um

$$R_2 - kQ = M + kQ - kQ = M$$

zu berechnen und erhält so die Kurvenversion M des Klartexts m .

Damit Coati, die die Nachricht abgefangen hat, zu M gelangen kann, müsste sie d aus $Q = dP$ oder k aus kQ bekommen, da sowohl P als auch Q , aber weder d noch k , öffentlich sind. Beides ist das diskrete Logarithmusproblem der elliptischen Kurve.

14.4 Die Kosten von Operationen mit elliptischen Kurven

Der Vorteil der Verwendung von elliptischen Kurven anstelle von Arithmetik modulo großen Primzahlen besteht darin, dass das gleiche Sicherheitsniveau mit kleineren Zahlen erreicht werden kann, auf denen Arithmetik ausgeführt wird. Der Nachteil ist, dass eine einzelne Gruppenoperation auf einer elliptischen Kurve viel komplizierter ist als die für RSA erforderliche Multiplikation modulo P . Es lohnt sich, die Gruppenoperationen zu betrachten, um zu überlegen, was es bedeutet, elliptische Kurven für die Kryptographie zu verwenden.

Wir werden die Arithmetik anhand der Weierstrass-Form einer Kurve veranschaulichen, die wir modulo N nehmen, und um das Problem der Inversionen modulo N (was jedes Mal eine ggT-Operation erfordern würde) zu vermeiden, werden wir die Jacobische Version der Kurve

$$\mathcal{E} : Y^2 = X^3 + aXZ^4 + bZ^6.$$

betrachten und Tripel (X, Y, Z) anstelle von Paaren (X, Y) verfolgen.

Wir erinnern uns auch daran, dass das Quadrieren einer Zahl weniger kostspielig ist als das Multiplizieren zweier Zahlen, daher werden wir das Quadrieren als etwas anderes als die Multiplikation betrachten.

14.4.1 Verdoppelung eines Punktes

Wenn unser Ziel darin besteht, einen Punkt $P = (X, Y, Z)$ zu verdoppeln, unter Verwendung von Jacobianischen Koordinaten, wie in [3] und anderswo beschrieben, können wir das Dreifache (x_3, y_3, z_3) mit sechs Quadraturen, vier Multiplikationen und einigen Additionen und Verschiebungen berechnen, die im Vergleich zu den Quadraturen und Multiplikationen kostengünstig sind, wie hier gezeigt.

$P_3 = (x_3, y_3, z_3)$ mit

$$\begin{aligned} m_{\text{num}} &= 3x_1^2 + az_1^4 \\ x_3 &= m_{\text{num}}^2 - 8x_1y_1^2 \\ y_3 &= m_{\text{num}}(4x_1y_1^2 - x_3) - 8y_1^4 \\ z_3 &= 2y_1z_1 \end{aligned}$$

in der Reihenfolge ausgeführt als

$A = y_1^2$	square
$B = 4x_1A$	multiply, shift
$C = 8A^2$	square, shift
$D = 3x_1^2 + az_1^4$	square, square, square, multiply, add
$x_3 = D^2 - 2B$	square, shift, add
$y_3 = D(B - x_3) - C$	add, multiply
$z_3 = 2y_1z_1$	multiply, shift

Wir stellen fest, dass einige dieser Berechnungen parallel durchgeführt werden können; dies könnte nicht relevant sein, wenn es in Software durchgeführt wird (obwohl Threading möglich sein könnte), aber es ist etwas, das in Betracht gezogen werden könnte, wenn man tatsächlich Hardware speziell für die Durchführung von elliptischer Kurvenarithmetik baut.

14.4.2 Von links nach rechts „Exponentiation“

Eine weitere Vereinfachung der Arithmetik in der elliptischen Kurvenkryptographie ergibt sich, wenn man sich die Algorithmen ansieht, die zur Berechnung von $m \cdot Q$ für einen Multiplikator m und einen Punkt Q auf der Kurve verwendet werden. Wir haben in Abschn. 6.4 über Exponentiation gesprochen, die durch Betrachtung der Bits des Exponenten erfolgt. Wir exponentieren nicht mit elliptischen Kurven, sondern berechnen den Punkt mQ für einen bestimmten Multiplikator m und einen Basispunkt Q . Die übliche Multiplikation von rechts nach links ist wie im hier gezeigten Python-Code.

```
# Compute P = m times point Q
# expanding the bits of m right to left.
P = identity
point_to_add_in = Q
while m != 0:
    if m % 2 == 1:
        P = P + point_to_add_in
    m = m // 2
    point_to_add_in = point_to_add_in + point_to_add_in
return P
```

Wir können jedoch das gleiche Ergebnis erzielen, indem wir die Bits von links nach rechts erweitern, wie im untenstehenden Code.

Die Methode von links nach rechts ist etwas mühsamer zu beginnen, da wir eine Potenz von zwei benötigen, um das linkeste 1-Bit des Exponenten zu finden. Im Algorithmus von rechts nach links müssen wir den Punkt verdoppeln, der für jedes Bit

des Exponenten hinzugefügt wird, und im Algorithmus von links nach rechts verdoppeln wir den laufenden Rückgabepunkt P . Es gibt keine Einsparungen dort. Die Einsparung besteht darin, dass wir, wenn das Bit eine 1 ist, den Basispunkt hinzufügen, so dass, wenn diese Kurvenarithmetik billiger ist als eine allgemeine Punktaddition, weil wir einen ausgezeichneten Punkt als unsere Basis gewählt haben, wir Zeit sparen.

```
# Compute P = m times point Q
# expanding the bits of m left to right.
P = identity
poweroftwo = A LARGE POWER OF TWO
while m != 0:
    P = P + P
    if power_of_two <= m:
        P = P + Q
        m = m - power_of_two
    power_of_two = power_of_two // 2
return P
```

Kurvenarithmetik ist von Natur aus teuer, daher ist es ein Gewinn in Bezug auf die Laufzeit, wenn wir das gleiche Sicherheitsniveau beibehalten können, aber den Basispunkt Q so wählen, dass die Addition billig ist.

14.4.2.1 Hinzufügen eines ausgezeichneten Punktes zu einem beliebigen Punkt

Das Hinzufügen von $P_1 = (x_1, y_1, z_1)$ zu einem ausgezeichneten „leichten“ Punkt $P_2 = (x_2, y_2, 1)$ kann folgendermaßen durchgeführt werden, um $P_3 = (x_3, y_3, z_3)$ zu erzeugen.

$$\begin{aligned}x_3 &= (y_2 z_1^3 - y_1)^2 - (x_2 z_1^2 - x_1)^2 (x_1 + x_2 z_1^2) \\y_3 &= (y_2 z_1^3 - y_1)(x_1(x_2 z_1^2 - x_1)^2 - x_3) - y_1(x_2 z_1^2 - x_1)^3 \\z_3 &= (x_2 z_1^2 - x_1)z_1\end{aligned}$$

Wie beim Verdoppeln können diese in einer effizienten Weise sequenziert werden, wie in [3] und anderswo beschrieben, wobei drei Quadrierungen, acht Multiplikationen und einige Verschiebungen und Additionen erforderlich sind.

$A = z_1^2$	square
$B = z_1 A$	multiply
$C = x_2 A$	multiply
$D = y_2 B$	multiply
$E = C - x_1$	add
$F = D - y_1$	add
$G = E^2$	square
$H = GE$	multiply
$I = x_1 G$	multiply
$x_3 = F^2 - H - 2I$	square, shift, add, add
$y_3 = F(I - x_3) - y_1 H$	add, multiply, multiply, add
$z_3 = z_1 E$	multiply

Im letzteren Fall von unterschiedlichen Punkten P_1 und P_2 können wir das Dreifache (x_3, y_3, z_3) mit zwei Quadrierungen, zwölf Multiplikationen, sieben Additionen und zwei Verschiebungen berechnen, wobei wir erneut darauf hinweisen, dass die Operationen innerhalb jedes Schrittes parallel ausgeführt werden können.

Es wurde umfangreiche Arbeit geleistet, um herauszufinden, wie diese Arithmetik am besten durchgeführt werden kann, da sie naiv durchgeführt teuer wäre.

14.5 Die Empfehlungen des NIST

Angesichts der Möglichkeiten, mit der für die öffentliche Schlüsselverschlüsselung benötigten Arithmetik zu spielen, ist es nicht überraschend, dass die von NIST empfohlenen elliptischen Kurven solche sind, bei denen die Arithmetikspiele gut funktionieren. Der NIST FIPS 186-4 (Federal Information Processing Standard), datiert Juli 2013, empfiehlt [4]

Bitlänge	Primkörper	Binärkörper
161 – 223	$P_{192} = 2^{192} - 2^{64} - 1$	$t^{163} + t^7 + t^6 + t^3 + 1$
224 – 255	$P_{224} = 2^{224} - 2^{96} + 1$	$t^{233} + t^{74} + 1$
246 – 383	$P_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$t^{283} + t^{12} + t^7 + t^5 + 1$
384 – 511	$P_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$	$t^{409} + t^{87} + 1$
> 511	$P_{521} = 2^{521} - 1$	$t^{571} + t^{10} + t^5 + t^2 + 1$

Im Falle der Primzahlenfelder sind die empfohlenen Kurven der Form

$$\mathcal{E} : y^2 = x^3 - 3x + b \pmod{P}$$

und zufällige Basispunkte werden empfohlen.

Im Falle der binären Felder sind die empfohlenen Kurven entweder der Form

$$\mathcal{E} : y^2 + xy = x^3 + x^2 + b$$

für vorgeschlagene Werte von b oder sind *Koblitz-Kurven*

$$\mathcal{E} : y^2 + xy = x^3 + ax^2 + 1$$

mit $a = 0$ oder 1 .

In allen Fällen sind die empfohlenen Basispunkte von einer großen Primordnung n und die Anzahl der Punkte auf der Kurve ist hn , für einen Kofaktor h , der $1, 2$ oder 4 ist.

Der FIPS 186-4 Standard kommt komplett mit den Spielen, die gespielt werden müssen, um effiziente Arithmetik zu ermöglichen, die einfache Verallgemeinerungen des verwendeten Arithmetikspiels sind, und in Kap. 8, wenn man beweist, dass eine Mersenne-Zahl prim ist. Zum Beispiel kann für das 192-Bit-Primzahlenfeld jede Zahl kleiner als P_{192}^2 (und somit das Ergebnis jedes Multiplikationsschritts) geschrieben werden als

$$A = A_5 \cdot 2^{320} + A_4 \cdot 2^{256} + A_3 \cdot 2^{192} + A_2 \cdot 2^{128} + A_1 \cdot 2^{64} + A_0$$

wo jeder der A_i eine 64-Bit-Zahl ist. Der Wert

$$B \equiv A \pmod{P_{192}}$$

kann berechnet werden als

$$B \equiv T + S_1 + S_2 + S_3 \pmod{P_{192}}$$

wo die 192-Bit-Summanden durch geeignetes Verketteten der A_i erzeugt werden:

$$T = A_2 || A_1 || A_0$$

$$S_1 = A_3 || A_3$$

$$S_2 = A_4 || A_4 || A_0$$

$$S_3 = A_5 || A_5 || A_5$$

Die modulare Reduktion wurde ersetzt durch Bitextraktion, Verkettung, Addition und vielleicht einige Subtraktionen des Moduls.

Die zur Reduktion modulo der anderen Primzahlen benötigte Arithmetik ist ebenfalls im FIPS vorhanden, und analoge Arithmetikspiele werden für die Arbeit in den binären Feldern vorgestellt.

14.6 Angriffe auf elliptische Kurven

Wir haben bereits festgestellt, dass der Indexkalkülansatz nicht funktioniert, um das diskrete Logarithmusproblem für elliptische Kurven anzugreifen, oder zumindest wurde der potenzielle Angriff mit Xedni nicht effektiv gemacht [5, 6].

Welche Angriffe werden dann verwendet?

14.6.1 Pohlig-Hellman Angriffe

Der offensichtlichste Angriff auf ein diskretes Logarithmusproblem einer elliptischen Kurve wäre der von Pohlig-Hellman [7]. Dieser Angriff ist jedoch nur effizient, wenn die Ordnung der Gruppe N eine glatte Zahl ist, da er in einer Zeit läuft, die \sqrt{N} ist, wobei N die Ordnung der Gruppe ist. Wenn die Gruppenordnung glatt ist, läuft Pohlig-Hellman, indem er die diskreten Logarithmen für jeden Faktor von N bestimmt und dann den gewünschten diskreten Logarithmus mit dem Chinesischen Restsatz erstellt. Dies ist unwahrscheinlich für kryptanalytische Probleme relevant zu sein, da ein professioneller Designer keine Kurven wählen würde, die diesem Angriff ausgesetzt sind. Es ist speziell, um einen solchen Angriff zu verhindern, dass die NIST-Kurven und alle von echten Profis gewählten Kurven einen großen Primfaktor haben, der die Ordnung der Gruppe teilt, und einen Kofaktor (in den NIST FIPS als h bezeichnet und so gewählt, dass $h = 1, 2, 4$) so klein wie möglich ist.

14.6.2 Pollard Rho Angriffe

Der Pollard Rho Angriff auf diskrete Logarithmusprobleme leitet sich vom Pollard Rho Faktorisierungsalgorithmus ab. Es handelt sich um einen $\mathcal{O}(\sqrt{P})$ Algorithmus, weshalb der Indexkalkül im Allgemeinen für diskrete Logarithmen modulo großer Primzahlen bevorzugt wird. In Ermangelung eines effektiven Indexkalküls für elliptische Kurven ist der beste Algorithmus für Kurven eine Variante von Pollard Rho.

14.6.2.1 Pollard Rho Modulo Primzahlen

Wenn wir Pollards ursprünglichem Ansatz für Logarithmen modulo Primzahlen folgen würden [8], beginnen wir mit einer primitiven Wurzel r und einem Wert q , für den wir den diskreten Logarithmus benötigen, und wir berechnen Sequenzen von Exponenten a_i und b_i und verwenden Sie diese zur Berechnung von

$$x_i \equiv q^{a_i} r^{b_i} \pmod{p}.$$

Wir möchten, dass die a_i und b_i einen mehr oder weniger zufälligen Durchlauf durch die Ganzzahlen modulo p liefern, und wählen

$$\begin{aligned}
a_{i+1} &\equiv a_i + 1 \pmod{p-1} && \text{if } 0 < x_i < p/3 \\
&\equiv 2a_i \pmod{p-1} && \text{if } p/3 < x_i < 2p/3 \\
&\equiv a_i \pmod{p-1} && \text{if } 2p/3 < x_i < p
\end{aligned}$$

und

$$\begin{aligned}
b_{i+1} &\equiv b_i \pmod{p-1} && \text{if } 0 < x_i < p/3 \\
&\equiv 2b_i \pmod{p-1} && \text{if } p/3 < x_i < 2p/3 \\
&\equiv b_i + 1 \pmod{p-1} && \text{if } 2p/3 < x_i < p
\end{aligned}$$

Wir können jetzt, praktisch ohne Speicher, x_i und x_{2i} berechnen, indem wir eine Sequenz einmal und die andere zweimal durchlaufen, die Werte von a_i und b_i im Auge behalten und nach einer Kollision suchen

$$x_i = x_{2i}.$$

Wenn wir eine Kollision bekommen, haben wir

$$q^{a_i} r^{b_i} \equiv q^{a_{2i}} r^{b_{2i}} \pmod{p}$$

und somit

$$q^m \equiv q^{a_i - a_{2i}} \equiv r^{b_{2i} - b_i} \equiv r^n \pmod{p}$$

Wir haben q zur m -ten Potenz als Potenz der primitiven Wurzel r ; wir wollen q zur ersten Potenz in Bezug auf r . Also lösen wir mit dem erweiterten euklidischen Algorithmus

$$g = \lambda m + \mu(p-1)$$

für λ und μ , und exponentieren Sie beide Seiten:

$$q^g \equiv (q^m)^\lambda \equiv (r^n)^\lambda \equiv r^{gk} \pmod{p}$$

Wir können n durch g teilen und $r^k, r^{2k}, r^{3k}, \dots$ ausprobieren, bis wir auf q stoßen. Da g klein sein sollte, ist dieser letzte Schritt nicht schwierig.

Lassen Sie uns ein Beispiel machen, mit $p = 31$, $r = 3$, und $q = 22$. Nur zur Referenz, wir werden die Potenzen von 3 auflisten.

Potenz	Wert	Potenz	Wert	Potenz	Wert
1	3	11	13	21	15
2	9	12	8	22	14
3	27	13	24	23	11
4	19	14	10	24	2
5	26	15	30	25	6
6	16	16	28	26	18

Potenz	Wert	Potenz	Wert	Potenz	Wert
7	17	17	22	27	23
8	20	18	4	28	7
9	29	19	12	29	21
10	25	20	5	30	1

Jetzt führen wir den Pollard-Algorithmus aus:

Index	a_i	q^{a_i}	b_i	r^{b_i}	$q^{a_i} r^{b_i}$
1	0	1	0	1	1
2	1	22	0	1	22
2	1	22	0	1	22
4	2	19	1	3	26
3	1	22	1	3	4
6	4	20	4	19	8
4	2	19	1	3	26
8	5	6	5	26	1
5	2	19	2	9	16
10	6	8	6	16	4
6	4	20	4	19	8
12	7	21	7	17	16
7	5	6	4	19	21
14	15	30	14	10	21

Wir bekommen die Kollision mit $i = 7$, $a_7 = 5$, $a_{14} = 15$, $b_7 = 4$, $b_{14} = 14$. Das gibt uns $m = 5 - 15 \equiv 20 \pmod{30}$ und $n = 14 - 4 \equiv 10 \pmod{30}$, und $22^{20} \equiv 3^{10} \equiv 25 \pmod{31}$. Wir erhalten $\lambda = 2$ und $g = 10$, und wir exponentieren

$$22^{10} \equiv 22^{40} \equiv 3^{20} \pmod{31}$$

Wir beginnen mit $3^2 \equiv 9 \pmod{31}$, weil $20/10 = 2$, und dann multiplizieren wir die 10-ten Einheitswurzeln modulo 31 ein; das wären $3^3, 3^6, 3^9, \dots, 3^{27}, 3^{30}$.

Wir erhalten $3^2 \equiv 9 \pmod{31}$, $3^5 \equiv 26 \pmod{31}$, $3^8 \equiv 20 \pmod{31}$, $3^{11} \equiv 13 \pmod{31}$, $3^{14} \equiv 10 \pmod{31}$, $3^{17} \equiv 22 \pmod{31}$, und wir sind fertig.

14.6.3 Pollard Rho für Kurven

Gegeben sind Punkte P und Q auf einer Kurve, das grundlegende Ziel ist es, unterschiedliche Paare von Paaren, (a, b) und (a', b') , zu finden, so dass

$$aP + bQ = a'P + b'Q$$

auf der Kurve. Wenn dies möglich ist, dann wissen wir, dass

$$(a - a')P = (b - b')Q = (b - b')dP$$

wo P und Q öffentliche Informationen sind, aber d ist der private Schlüssel. Dies würde bedeuten, dass

$$(a - a') \equiv (b - b')d \pmod{n}$$

wo n die Ordnung des Punktes P auf der Kurve ist. Da n bekannt ist, können wir $b - b'$ umkehren, um

$$d \equiv (a - a')(b - b')^{-1} \pmod{n}$$

Die Pollard-Rho-Version des Angriffs besteht darin, eine Zufallsfunktion für das Durchlaufen der Kurve zu wählen. Anstatt nur einmal und zweimal zu gehen, wie im Faktorisierungsalgorithmus, gehen wir mit unterschiedlichen Geschwindigkeiten vor (und dies kann sehr effektiv parallelisiert werden), bis wir die erwartete Kollision bekommen. Wenn wir die Kollision bekommen, mit

$$aP + bQ = a'P + b'Q$$

für $a \neq a'$ und $b \neq b'$, dann werden wir unsere Lösung bekommen. Dies läuft, wie auch Pollard Rho für Faktorisierung, in einer Zeit proportional zur Quadratwurzel der Ordnung der Gruppe.

14.6.4 Pollard Rho parallel

Der vorgestellte Algorithmus ist ein einzelner Pfad durch das Rho-Diagramm. Aber wir stellen fest, dass sobald eine Kollision gefunden wird, nichts im Rest des Algorithmus davon abhängt, dass die Kollision für bestimmte Indizes i und $2i$ gefunden wurde. Wir benötigen die Tripel a_i , b_i , und $q^{a_i}r^{b_i}$, für zwei Indizes, die zu einer Kollision führen, aber es wird kein Gebrauch von dem Wert i gemacht.

Wenn wir all den Speicher der Welt hätten und einen schnellen Suchbaum, könnten wir eine Reihe von parallelen Schritten von verschiedenen Anfangsindizes starten, die a_i , b_i , Paare mit einem Index von $q^{a_i}r^{b_i}$ speichern und dann nach Kollisionen im Baum suchen könnten.

Besser noch, obwohl wir von einem Suchbaum profitieren würden, brauchen wir ihn nicht, wenn wir nur die Existenz einer Kollision feststellen wollen. Wir müssen die Tripel a_i , b_i , und $q^{a_i}r^{b_i}$ speichern, aber zur Erkennung von Kollisionen benötigen wir eine gute Hash-Funktion, die $q^{a_i}r^{b_i}$ als Eingabe verwendet. Wir werden sehr lange parallel rechnen und nur sehr selten Kollisionen bekommen (es gibt einige Kollisionen, die nutzlos sein könnten, zum Beispiel, wenn wir irgendwie auf die gleichen a_i , b_i , Werte

stoßen). Das Speichern der Tripel ist ein Backend-Prozess. Die Überprüfung auf $q^{a_i} r^{b_i}$ Kollisionen geschieht ständig und muss so effizient wie möglich sein, und da Kollisionen selten sind, können wir es uns leisten, ein wenig mehr zu suchen, wenn wir glauben, etwas gefunden zu haben.

Wenn der Zyklus groß ist, wie es in der Kryptographie für Primzahlen oder für elliptische Kurven der Fall wäre, ist das Hindernis hier der Bedarf an enormen Speicherressourcen. Das kann etwas gemildert werden, indem man sich dafür entscheidet, nicht alle Schritte zu speichern, sondern nur eine Teilmenge, vielleicht nur diejenigen, für die $q^{a_i} r^{b_i}$ „klein“ ist. Dies wird dann zu einer Standard-Computational-Balancing-Akt:

- Wie viele Prozessoren haben wir?
- Wie viel Speicher haben wir zum Speichern von Paaren?
- Wie schnell können wir Werte mit einer Hash-Funktion nachschlagen?
- Wie balancieren wir die Rate des Steppings mit den Kosten der Kollisionssuche mit den Kosten des Mehraufwands an Rechenleistung, weil wir nicht genug Speicher haben?

Wie bei allem in den Berechnungen, die für das Faktorisieren, für die Index-Kalkulation oder für den Pollard-Rho-Angriff auf ein elliptisches Kurven-Discrete-Log gemacht werden, ist der effizienteste Ansatz ein bewegliches Ziel, das teilweise auf den verfügbaren Rechenressourcen und der Natur der zugrunde liegenden Hardware und Systemsoftware basiert. Die Kunst (und nicht Wissenschaft) der Durchführung der Berechnungen auf handelsüblichen Computern wäre sehr unterschiedlich von dem, was der beste Ansatz wäre, wenn man FPGAs oder GPUs oder spezielle Hardware entwerfen würde.

14.7 Ein Vergleich der Komplexitäten

Ein Weißbuch, das von der Atmel Corporation veröffentlicht wurde, vergleicht die kryptographische Sicherheit verschiedener Kryptosysteme auf der Grundlage der Bitlängen der Operanden. Ihre Ergebnisse [9] zitieren ein früheres NSA-Dokument und sind in Tab. 14.1. Obwohl diese sich je nach der Rechenleistung verschiedener Arten von Hardwareplattformen (Supercomputer, Desktops, Computer mit FPGAs oder GPUs usw.) verschieben können, bietet diese Tabelle einen Ausgangspunkt für den Vergleich.

Tab. 14.1 Sicherheit-AES versus RSA versus elliptische Kurven

Sicherheitsbits	AES	Mindestbits RSA	Mindestbits ECC
128	AES-128	3072	256
192	AES-192	7680	384
256	AES-256	15360	512

14.8 Übungen

1. (Programmierübung.) Implementieren Sie eine allgemeine Klasse für elliptische Kurvenarithmetik modulo einer Primzahl, unter Verwendung von Jacobischen Koordinaten, aber mit einer Reduktion auf den Punkt mit $z = 1$, damit Sie mit weniger Aufwand sehen können, dass Sie die Arithmetik richtig gemacht haben.
2. (Programmierübung.) Implementieren Sie einen elliptischen Kurvenschlüsselaustausch mit dem Basispunkt $Q = (1, 6)$ auf der Kurve $x^3 + x + 3$ genommen modulo 31. Diese Kurve hat einen Zyklus von Primlänge 41.
3. (Programmierübung.) Implementieren Sie ein kryptographisches System mit elliptischen Kurven unter Verwendung der Kurve $x^3 + x + 3$ genommen modulo 31. Diese Kurve hat einen Zyklus von Primlänge 41, mit einem Basispunkt $P = (1, 6)$.

Literatur

1. N. Koblitz, Elliptic curve cryptosystems. *Math. Comput.* **48**, 203–209 (1987)
2. V.S. Miller, Use of elliptic curves in cryptography, in *Advances in cryptology – CRYPTO '85*, Bd. 218, Lecture notes in computer science (1986), S. 417–426
3. D. Hankerson, A. Menezes, S. Vanstone, *Guide to elliptic curve cryptography* (Springer, 2004)
4. NIST, *Fips 186-4: digital signature standard* (2013). <https://csrc.nist.gov/publications/detail/fips/186/4/final>
5. M.J. Jacobson, N. Koblitz, J.H. Silverman, A. Stein, E. Teske, Analysis of the Xedni calculus attack. *Des. Codes Cryptogr.* **20**, 41–64 (2000)
6. J.H. Silverman, The Xedni calculus and the elliptic curve discrete logarithm problem. *Des. Codes Cryptogr.* **20**, 5–40 (2000)
7. S. Pohlig, M. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Trans. Inf. Theory* **24**, 106–110 (1978)
8. J.M. Pollard, Monte Carlo methods for index computation mod p . *Math. Comput.* 918–924 (1978)
9. K. Maletski, *RSA vs ECC comparison for embedded systems* (Atmel Corporation white paper, 2015)

Zusammenfassung

Mit der Veröffentlichung von Peter Shors bahnbrechendem Artikel, dass das Faktorisieren und diskrete Logarithmenberechnungen auf einem Quantencomputer völlig machbar wären, und mit Fortschritten im Bau von Quantencomputern, hat sich der Fokus auf das sogenannte „Post-Quanten-Kryptographie“ gerichtet. Zu den vielversprechendsten Kandidaten für die Post-Quanten-Kryptographie gehören Kryptosysteme, die auf dem Problem der Suche nach kurzen Vektoren in Gittern basieren. In diesem Kapitel skizzieren wir kurz, warum Quantencomputer RSA-artige Kryptosysteme obsolet machen können und wie Gitter in der Kryptographie verwendet werden können. Wir konzentrieren uns auf das vielleicht bekannteste Gittersystem, NTRU, und erklären, wie es verwendet wird und warum Angriffe darauf immer noch rechnerisch unpraktikabel erscheinen.

15.1 Quantencomputing

Um Physiker zu werden, müssen Sie in Blut unterschreiben, dass Sie sich nicht über Dinge aufregen, die keinen Sinn ergeben und nicht erklärt werden können. (Ed Fredkin [1]) Sehr informell kann man sagen, dass die Stärke eines Quantencomputers darin besteht, alles auf einmal berechnen zu können. Das ist nicht ganz richtig, aber es gibt einen Einblick, wie ein Quantencomputer RSA-Typ-Faktorisierungs- und Diffie-Hellman-Typ-Diskrete-Logarithmus-Probleme angreifen kann. Ein Quantencomputer befindet sich zu jedem Zeitpunkt der Berechnung in einer Überlagerung von Zuständen

$$\sum_i a_i |S_i\rangle$$

wo jeder $|S_i\rangle$ ein Zustand des Computers ist und die a_i komplexe Zahlen sind, die die Amplituden darstellen. Die Quantenphysik besagt, dass die Untersuchung des Systems es verändert, aber wenn der Computer zu irgendeinem Zeitpunkt untersucht wird, ist die Wahrscheinlichkeit, den Zustand $|S_i\rangle$ zu beobachten, $|a_i|^2$, und dies sind tatsächlich Wahrscheinlichkeiten: wir haben $\sum_i |a_i|^2 = 1$.

Obwohl voll funktionsfähige, allgemein einsetzbare Quantencomputer noch nicht gebaut wurden, werden Fortschritte gemacht, und es wird wahrscheinlich einen Punkt in der nicht allzu fernen Zukunft geben, an dem Quantencomputer für reale Rechenprobleme eingesetzt werden können. Eine der bedeutendsten Konsequenzen der Fähigkeit, einen echten Quantencomputer zu bauen, ist, dass alle auf Zahlentheorie basierenden öffentlichen Schlüsselkryptographien, wie sie in den Kap. 10–14 vorgestellt wurden, angreifbar sein werden und keine Sicherheit mehr bieten. Dies war das Hauptergebnis einer bahnbrechenden Arbeit, die 1994 von Peter Shor [2–4] veröffentlicht wurde. Seit der Veröffentlichung dieser Arbeit, und insbesondere in den letzten Jahren, da die Fortschritte beim Bau echter Quantencomputer zugenommen haben, hat sich der Fokus auf kryptographische Systeme verlagert, die nicht anfällig für Angriffe auf Quantencomputer sind.

Wir werden nur einen Überblick über die Faktorisierung mit einem Quantencomputer geben, anstatt die Details, die vielleicht mehr Mathematik und mehr Physik erfordern, als notwendig ist, um die Notwendigkeit der Entwicklung alternativer Kryptosysteme zu verstehen.

Wir erinnern uns daran, dass das RSA-Kryptosystem schwer anzugreifen ist, weil es auf Arithmetik modulo $N = pq$ für zwei große Primzahlen p und q basiert, weil der Angriff auf das System das Wissen (oder zumindest scheint es derzeit so) der Euler-Phi-Funktion $\phi(N) = (p-1)(q-1)$ erfordert, und weil das Berechnen von $\phi(N)$ genauso schwierig zu sein scheint wie das Faktorisieren. Die Siebmethoden zum Faktorisieren basieren auf der Suche nach X und Y , so dass

$$X^2 - Y^2 = (X + Y)(X - Y) \equiv 0 \pmod{N}$$

und das Finden von ausreichend vielen solchen X und Y , dass wir schließlich $\gcd(X + Y, N)$ und $\gcd(X - Y, N)$ haben, die jeweils eine von p und q enthalten, aber nicht beide zusammen in einem der ggT multipliziert.

Wenn wir ein X erzeugen können, so dass $X^2 - 1 \equiv 0 \pmod{N}$ durch das Finden einer nichttrivialen Quadratwurzel von 1, dann können wir wahrscheinlich N faktorisieren.

Als Beispiel faktorisieren wir $1457 = 31 \times 47$. Wir haben $\phi(1457) = 1380 = 2^2 \times 3 \times 5 \times 23$; der Teil modulo 31 ist bröckelig, aber 47 ist zweimal eine Primzahl plus 1.

Die nichttrivialen Quadratwurzeln von 1 modulo 31 und 47 sind natürlich 30 und 46. Wir können direkt berechnen (weil dies ein kleines Beispiel ist), dass die nichttrivialen Quadratwurzeln von 1 modulo 1457 $187 = 6 \times 31 + 1 = 4 \times 47 - 1$ und $1270 = 41 \times 31 - 1 = 27 \times 47 + 1$ sind; die vier Quadratwurzeln von 1 modulo das Produkt von zwei Primzahlen werden das Muster von $(1, 1)$, $(-1, 1)$, $(1, -1)$, und $(-1, -1)$, modulo den beiden Primzahlen haben.

Genauer gesagt, können wir die Ordnungen der Reste modulo 1457 berechnen. Wir wissen, dass die maximale Ordnung 690 ist, und wir finden, dass 3 die Ordnung 690 hat. Wenn wir einen Faktor von 2 in der Exponenten zurücknehmen, was das Quadratwurzelziehen bedeutet, finden wir

$$\gcd(3^{345} + 1 \pmod{1457}, 1457) = 31$$

und

$$\gcd(3^{345} - 1 \pmod{1457}, 1457) = 47.$$

Auf einem Standardcomputer wäre es ineffizient, zufällige Reste auszuwählen und die Ordnung zu berechnen, da wir keine gute Möglichkeit hätten, einen Rest hochzuzählen und alle Potenzen zu überprüfen, um den einen Wert zu finden, der zufällig 1 ist. Mit einem Quantencomputer jedoch ermöglicht uns die Überlagerung von Zuständen im Wesentlichen, alle Potenzen gleichzeitig zu überprüfen. Durch die Durchführung einer Quanten-Fourier-Transformation (die wir nicht erklären werden), erhalten wir Näherungswerte für die Ordnung und können in polynomialer Zeit feststellen, ob die Näherung ausreichend genau ist, um uns die genaue Ordnung bestimmen zu lassen. Da wir die Ordnung eines zufällig ausgewählten Restes in Quantenpolynomialzeit bestimmen können, können wir in Polynomialzeit einen Faktor von N finden.

15.2 Gitter: Eine Einführung

Definition 15.1 Wir betrachten Zeilenvektoren \mathbf{b}_i , für $i = 1, \dots, n$, die eine Länge von n haben, Koeffizienten in den reellen Zahlen \mathbb{R} haben und über \mathbb{R}^n linear unabhängig sind. Ein *Gitter* über der Basis $\{\mathbf{b}_i\}$ ist die Menge

$$L(\mathbf{b}_1, \dots, \mathbf{b}_n) = \left\{ \sum_{i=1}^n x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}$$

Offensichtlich können wir dies in der Sprache der Matrizen formulieren. Wenn wir schreiben

$$\mathbf{B} = \begin{pmatrix} \mathbf{b}_1 \\ \dots \\ \mathbf{b}_n \end{pmatrix}$$

dann kann das Gitter als die Menge der Matrixprodukte geschrieben werden

$$L(\mathbf{B}) = \{\mathbf{x}\mathbf{B} : \mathbf{x} \in \mathbb{Z}^n\}$$

Wir beobachten, dass wenn \mathbf{U} eine Matrix mit ganzzahligen Koeffizienten und Determinante ± 1 ist, dann existiert eine inverse Matrix \mathbf{U}^{-1} deren Koeffizienten ebenfalls ganze Zahlen sind, und wir haben, dass

$$L(\mathbf{B}) = L(\mathbf{UB}).$$

Die Matrix \mathbf{U} ist eine Basiswechsellmatrix, und einige Matrizen \mathbf{U} können durch Matrixreduktion auf \mathbf{B} . erhalten werden. Tatsächlich, wenn wir eine der Standard-Matrixreduktionsschritte anwenden

- Tauschen Sie zwei Zeilen von \mathbf{B} aus.
- Multiplizieren Sie alle Einträge in einer Zeile mit -1
- Fügen Sie ein ganzzahliges Vielfaches einer Zeile zu einer anderen Zeile hinzu.

und führen Sie die Operationen auf der Matrix von k Zeilen und $2k$ Spalten \mathbf{BI} von \mathbf{B} mit einer Identität auf der rechten Seite durch, dann wird die Identitätsmatrix in die Matrix umgewandelt, die den Basiswechsel durchführt. Das einzige, was dies von der gewöhnlichen Gaußschen Elimination unterscheidet, ist, dass wir darauf achten müssen, nicht durch Werte außer ± 1 zu multiplizieren oder zu teilen, weil wir die Matrixkoeffizienten als ganze Zahlen beibehalten müssen.

Wir können die Reduktion von \mathbf{B} zu einer Basis mit mehr Einschränkungen mit einem einfachen Beispiel veranschaulichen.

Beispiel 15.1 Nehmen wir an, wir haben ein Gitter in drei Dimensionen, das durch

$$\mathbf{B} = \begin{pmatrix} 1 & 4 & 9 \\ 2 & 7 & 2 \\ 3 & 9 & 5 \end{pmatrix}$$

gegeben ist. Wir können die Matrix mit der Identität erweitern, um

$$\begin{pmatrix} 1 & 4 & 9 & X & 1 & 0 & 0 \\ 2 & 7 & 2 & X & 0 & 1 & 0 \\ 3 & 9 & 5 & X & 0 & 0 & 1 \end{pmatrix}$$

zu erhalten, wobei das X lediglich ein Marker ist, um die beiden Matrizen zu trennen.

Wir reduzieren in der ersten Spalte:

$$\begin{pmatrix} 1 & 4 & 9 & X & 1 & 0 & 0 \\ 0 & -1 & -16 & X & -2 & 1 & 0 \\ 0 & -3 & -22 & X & -3 & 0 & 1 \end{pmatrix}$$

Dann können wir in der zweiten Spalte reduzieren:

$$\begin{pmatrix} 1 & 4 & 9 & X & 1 & 0 & 0 \\ 0 & 1 & 16 & X & 2 & -1 & 0 \\ 0 & 0 & 26 & X & 3 & -3 & 1 \end{pmatrix}$$

An diesem Punkt können wir beobachten, dass

$$\mathbf{U} \cdot \mathbf{B} = \mathbf{B}' = \begin{pmatrix} 1 & 0 & 0 \\ 2 & -1 & 0 \\ 3 & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 4 & 9 \\ 2 & 7 & 2 \\ 3 & 9 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 9 \\ 0 & 1 & 16 \\ 0 & 0 & 26 \end{pmatrix}$$

und wir haben eine Basiswechselmatrix \mathbf{U} , die \mathbf{B} zu \mathbf{B}' überführt.

Wir stellen fest, dass diese Matrix ihre eigene Inverse ist.

15.3 Harte Gitterprobleme

Kryptographie, die auf Gittern basiert, stützt sich hauptsächlich (aber nicht ausschließlich) auf die Tatsache, dass es zwei Gitterprobleme gibt, die rechnerisch sehr schwer zu lösen sind.

Problem 1. (Kürzestes Vektorproblem) Gegeben ist ein Gitter L , was ist der kürzeste Vektor im Gitter? Das heißt, unter allen Vektoren $\mathbf{v} = \mathbf{x}\mathbf{B}$, für $\mathbf{x} \in \mathbb{Z}^n$, was ist der Vektor $\mathbf{v} = (v_1, \dots, v_n)$ dessen Länge

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^n v_i^2}$$

minimal ist?

Problem 2. (Nächster Vektorproblem) Gegeben ist ein Gitter L , und ein Punkt $\mathbf{r} \in \mathbb{R}^n$, was ist der Gittervektor, der am nächsten zu \mathbf{r} liegt?

Dies sind sehr ähnliche Probleme. Beide wurden ausführlich untersucht, und es gibt Algorithmen, die Annäherungen liefern, aber es gibt auch Beweise dafür, dass diese Probleme unter geeigneten Bedingungen in der Praxis rechnerisch unlösbar sind.

Beispiel 15.2 In unserem obigen Beispiel können wir feststellen (weil die Dimension klein ist), dass der kürzeste Vektor im Gitter

$$\begin{pmatrix} 1 & 3 & -3 \end{pmatrix} \begin{pmatrix} 1 & 4 & 9 \\ 2 & 7 & 2 \\ 3 & 9 & 5 \end{pmatrix} = \begin{pmatrix} -2 & -2 & 0 \end{pmatrix}$$

der Länge $\sqrt{8}$ ist.

Wir können auch überprüfen, dass

$$\begin{pmatrix} -2 & 6 & -3 \end{pmatrix} \begin{pmatrix} 1 & 4 & 9 \\ 0 & 1 & 16 \\ 0 & 0 & 26 \end{pmatrix} = \begin{pmatrix} -2 & -2 & 0 \end{pmatrix}$$

und dass

$$(-2 \ 6 \ -3) \begin{pmatrix} 1 & 0 & 0 \\ 2 & -1 & 0 \\ 3 & -3 & 1 \end{pmatrix} = (1 \ 3 \ -3)$$

wie es sein muss: wenn $\mathbf{B}' = \mathbf{UB}$, und $\mathbf{x}'\mathbf{B}' = \mathbf{x}\mathbf{B}$, dann müssen wir haben $\mathbf{x}'\mathbf{U} = \mathbf{x}$.

15.4 NTRU

Obwohl es auf den ersten Blick nicht offensichtlich sein mag, ist das von Hoffstein, Pipher und Silverman vorgeschlagene NTRU-Kryptosystem¹ [5] ein gitterbasiertes Kryptosystem. Wir betrachten den Ring R der Polynome $f(X)$, mit ganzzahligen Koeffizienten, reduziert modulo $X^N - 1$. Der Wert von N , dem Grad des Modulus, wird einer der Faktoren sein, die die Sicherheit des Kryptosystems beeinflussen. Die Addition von Elementen in diesem Ring ist die übliche Polynomaddition (wobei wir in diesem Fall die Koeffizienten *nicht* modulo 2 reduzieren). Die Multiplikation ist die übliche Multiplikation und anschließende Reduktion. Die Tatsache, dass wir $X^N \equiv 1$ haben, bedeutet, dass wir die Multiplikation als Faltungsprodukt schreiben können

$$h(X) = f(X) \circledast g(X)$$

wo die Koeffizienten h_k sind

$$h_k = \sum_{i+j \equiv k \pmod{N}} f_i \cdot g_j$$

Wir stellen fest, dass das Faltungsprodukt kommutativ ist.

Beispiel 15.3 Wenn wir $X^3 - 1$ als unseren Modulus und

$$f(X) = 1 + 2X + 3X^2$$

$$g(X) = 2 + 5X + 7X^2$$

dann

$$\begin{aligned} f(X) \circledast g(X) &= 2 + 5X + 7X^2 + 4X + 10X^2 + 14X^3 + 6X^2 + 15X^3 + 21X^4 \\ &= (2 + 14 + 15) + (5 + 4 + 21)X + (7 + 10 + 6)X^2 \\ &= 31 + 30X + 23X^2. \end{aligned}$$

¹ Wir werden diese Autoren im Text hier als „HPS“ bezeichnen.

Schließlich werden wir die Notation

$$f(X) \equiv g(X) \pmod{X^N - 1, q}$$

verwenden, um anzugeben, dass wir das Polynom $g(X)$ erhalten würden, indem wir $f(X)$ modulo $X^N - 1$ als Polynom reduzieren und dann die Koeffizienten des Polynoms modulo einer ganzen Zahl q reduzieren.

15.5 Das NTRU-Kryptosystem

Ausführliche Beschreibungen von NTRU finden sich in der Originalarbeit von Hoffstein, Pipher und Silverman [5] und in dem späteren Buch der gleichen Autoren [6].

15.5.1 Parameter

Eine gegebene Instanz von NTRU hat drei Ganzzahlparameter, den Grad N des als Modulus verwendeten Polynoms und zwei Ganzzahlen p und q ; und vier Mengen von Polynomen L_f , L_g , L_r , und L_m von Graden $N - 1$ mit ganzzahligen Koeffizienten. Wir benötigen eigentlich nicht, dass p und q prim sind, nur dass sie teilerfremd mit $\gcd(p, q) = 1$ sind, aber wir werden sie oft als prim nehmen, und wir werden q viel größer als p nehmen, aus Gründen, die später erscheinen werden.

Arbeiten im oben definierten Ring R , schreiben wir Polynome oder Vektoren

$$F = \sum_{i=0}^{N-1} F_i x^i = [F_0, \dots, F_{N-1}]$$

mit der Sequenz der Koeffizienten, die zu steigenden Potenzen von x gehören.

Wir definieren ein Gitter durch

$$L(d_1, d_2) = \left\{ F \in R : F \text{ has } \begin{cases} d_1 \text{ coefficients that are } 1, \\ d_2 \text{ coefficients that are } -1, \\ \text{and all other coefficients are } 0 \end{cases} \right\}$$

Wir verwenden d_f , d_g , und d als Ganzzahlbegrenzungen für die Anzahl der Nichtnull-Koeffizienten der Polynome f , g , r in R , und wir definieren Gitter

$$L_f = L(d_f, d_f - 1)$$

$$L_g = L(d_g, d_g)$$

$$L_r = L(d, d)$$

Wir wählen absichtlich $L_f = L(d_f, d_f - 1)$ so, dass f invertierbar ist.

15.5.2 Schlüssel erstellen

NTRU ist ein asymmetrisches, öffentliches Schlüssel-Kryptosystem. Um einen Schlüssel zu erstellen, wählt Armadillo zufällige Polynome f und g aus L_f und L_g aus, jeweils. Das Polynom f muss invertierbar modulo sowohl p als auch q sein, und wir bezeichnen seine Inversen durch F_p und F_q . Mit „Inversen“ meinen wir, dass

$$F_p \circledast f \equiv 1 \pmod{x^N - 1, p}$$

$$F_q \circledast f \equiv 1 \pmod{x^N - 1, q}$$

Wir stellen fest, dass die Berechnung der Inversen eine einmalige Arbeit vor der Einrichtung eines kryptographischen Systems ist und die Inversen zur späteren Verwendung gespeichert werden könnten.

Armadillo erstellt dann ihren öffentlichen Schlüssel

$$h = F_q \circledast g \pmod{q}$$

behält aber f (und F_p , wenn es gespeichert wird) privat.

15.5.3 Eine Nachricht verschlüsseln

Gegeben eine zu sendende Nachricht, schreibt Bobcat die Nachricht als Polynom m des Grades $N - 1$ mit Koeffizienten im Bereich $-\frac{p}{2}$ bis $\frac{p}{2}$. Bobcat wählt dann zufällig ein Polynom $r \in L_r$ als Einmal-Schlüssel, berechnet den Geheimtext²

$$e = p \cdot r \circledast h + m \pmod{q}.$$

und sendet e an Armadillo.

15.5.4 Eine Nachricht entschlüsseln

Nachdem sie den Geheimtext e erhalten hat, berechnet Armadillo, die ihren privaten Schlüssel f kennt,

$$a \equiv f \circledast e \pmod{q}$$

²Wir stellen fest, dass einige Darstellungen die Multiplikation mit p als Teil des öffentlichen Schlüssels h einschließen, während die HPS-Darstellung mit p multipliziert, wenn der Geheimtext erzeugt wird.

und reduziert die Koeffizienten a_i um die Grenzen zu erfüllen

$$\frac{-q}{2} \leq a_i < \frac{q}{2}.$$

Mit diesem Schritt stellt Armadillo m wieder her als

$$m \equiv F_p \circledast a \pmod{p}. \quad (15.1)$$

Beispiel 15.4 Schauen wir uns ein Beispiel an. Wir wählen Parameter $N = 5$, so reduzieren wir alle Polynome modulo $X^5 - 1$, das kleine Modul $p = 3$, und das große Modul $q = 17$. Wir wählen $d_f = 2, d_g = 2, d_r = 1$; Dies sind die Grenzen für die Anzahl der Nicht-Null-Koeffizienten der Polynome.

Mit diesem können wir wählen

$$f = -1 + X^2 + X^4 = [-1, 0, 1, 0, 1]$$

und

$$g = -1 + X + X^2 - X^3 = [-1, 1, 0, 1, -1]$$

und berechnen

$$F_p = -1 - X + X^3 - X^4 = [-1, -1, 0, 1, -1]$$

und

$$F_q = 6 + 5X + 8X^2 + 2X^3 + 14X^4 = [6, 5, 8, 2, 14]$$

und beachten, dass die Berechnung von F_p und von F_q einmalige Arbeit ist, die im Wesentlichen eine GCD-Operation in den entsprechenden Ringen ist.

Wir können nun den öffentlichen Schlüssel berechnen

$$h = 11 + 12X + 9X^2 + 15X^3 + 4X^4 = [11, 12, 9, 15, 4]$$

Nehmen wir nun an, die Nachricht ist das Polynom/Vektor

$$m = [-1, 0, 1, 1, 0]$$

wo wir den Vektor als steigende Potenzen von X nehmen.

Wir wählen zufällig $r = X - X^4$ und berechnen den Geheimtext

$$e = p \cdot h \circledast r + m \pmod{q},$$

was ergibt

$$e = 8 + 2X + 6X^2 + 8X^3 + 11X^4 = [8, 2, 6, 8, 11] \pmod{q},$$

Zur Entschlüsselung ist es notwendig zu berechnen

$$a' \equiv f \circledast e \equiv [2, -2, -7, 5, 3] \pmod{q}.$$

Wir berechnen dann

$$F_p \circledast a \equiv [-1, 0, 1, 1, 0] \pmod{p}.$$

und wir stellen die ursprüngliche Nachricht modulo p wieder her.

15.5.5 Warum das funktioniert

Der erste Schritt bei der Entschlüsselung besteht darin, e von links mit f zu multiplizieren, was ergibt

$$\begin{aligned} a' &= f \circledast e \\ &\equiv f \circledast pr \circledast h + f \circledast m \pmod{q} \\ &\equiv f \circledast pr \circledast F_q \circledast g + f \circledast m \pmod{q} \\ &\equiv pr \circledast g + f \circledast m \pmod{q} \end{aligned} \tag{15.2}$$

Wir reduzieren nun a' zu a , wobei alle Koeffizienten im Intervall $[-q/2, q/2)$ liegen. Wenn wir p und q richtig gewählt haben, erzeugt die Reduktion modulo q die genauen Koeffizienten dieses letzten Polynoms, nicht nur die Koeffizienten modulo q , so dass wir bei der Reduktion dieses letzten modulo p das genaue Polynom zurückbekommen

$$f \circledast m$$

und können durch Multiplikation mit F_p modulo p den Klartext m zurückbekommen.

15.5.6 Fehler bei der Entschlüsselung verhindern

Normalerweise würden wir nicht erwarten, dass die Reduktion modulo p und modulo q ein eindeutiges Ergebnis liefert. Die Argumentation, dass dieser Prozess eindeutig ist, läuft wie folgt ab.

Wir erinnern uns daran, dass die Koeffizienten von f , g und r nur $0, 1, -1$ gewählt wurden. Wir haben p viel kleiner als q gewählt, und wir können die Koeffizienten von m klein wählen, da sie eine Kodierung des Klartexts darstellen.

Das bedeutet, dass die *tatsächlichen ganzzahligen* Koeffizienten von

$$pr \circledast g + f \circledast m$$

wahrscheinlich klein sind und die Reduktion modulo q in den Bereich $-q/2$ bis $q/2$ möglicherweise nicht tatsächlich benötigt wird. Die Reduktion dieses Polynoms modulo p ergibt $f \circledast m$ und dann ergibt die Multiplikation mit F_p modulo p m .

Man kann eine Grenze beweisen; das Folgende ist Proposition 6.48 aus [6].

Satz 15.1 Wenn die Parameter N, p, q, d so gewählt werden, dass

$$q > (6d + 1)p$$

dann ist das von Armadillo berechnete Polynom gleich der Klartextnachricht m von Bobcat, wobei wir annehmen, dass $d_f - 1 = d_g = d_r = d$.

Beweis Wir wissen aus Gl. (15.2) dass

$$a' \equiv pr \circledast g + f \circledast m \pmod{q}. \quad (15.3)$$

Die Polynome r und g haben jeweils d Koeffizienten, deren Werte entweder $+1$ oder -1 sind. Wenn alle Koeffizienten übereinstimmen, dann wird der größte Koeffizient im Faltungsprodukt $r \circledast g$ $2d$ sein. Ebenso wird der größtmögliche Koeffizient im zweiten Term $f \circledast m$ $p(2d + 1)/2$ sein, so dass der größtmögliche Koeffizient von (15.3) ist

$$p(2d) + p(2d + 1)/2 = p\left(3d + \frac{1}{2}\right) < q/2.$$

Wenn wir q wie in der Annahme des Theorems gewählt haben, ist dieser letzte Wert kleiner als $q/2$ und die berechneten Koeffizienten modulo q als Reste symmetrisch um 0 sind die gleichen wie die Koeffizienten, die wir erhalten würden, wenn wir die Koeffizienten als gewöhnliche Ganzzahlen berechnen würden. Die Reduktion modulo p von a' zu a muss dann zur Wiederherstellung des ursprünglichen Nachrichtenpolynoms m führen. \square

Satz 15.1 ist ein Worst-Case-Ergebnis. Da die Polynome spärlich sind und ihre Koeffizienten klein, ist es unwahrscheinlich, dass man normalerweise auf diesen Worst Case trifft. Wir stellen fest, dass in unserem Beispiel, mit $p = 3$ und $d = 2$, diese Grenze erfordern würde, dass q größer als 17 ist, und doch erholen wir uns mindestens diese eine Nachricht mit $q = 17$. Diese Grenze ist jedoch nicht so übermäßig, dass man sie nicht in der Praxis verwenden könnte, und tatsächlich wurde dies getan.

15.6 Gitterangriffe auf NTRU

Es mag nicht offensichtlich sein, dass NTRU als ein gitterbasiertes Kryptosystem betrachtet werden kann; dies wird einfacher zu sehen, wenn man sich verschiedene Angriffe auf NTRU ansieht. Dieser Angriff stammt aus der Arbeit von Coppersmith und Shamir [7]. Wir werden zuerst ein Beispiel machen und dann die Theorie betrachten.

Wir lassen $(N, p, q) = (7, 3, 19)$, mit

$$\begin{aligned} f &= 1 - x + x^4 = [1, -1, 0, 0, 1, 0, 0] \\ g &= -1 - x + x^3 - x^5 = [-1, 1, 0, 1, 0, -1, 0] \end{aligned}$$

und wir erhalten den öffentlichen Schlüssel

$$h = [2, 18, 14, 6, 4, 4, 9]$$

als Koeffizienten von zunehmenden Potenzen von x .

Wir erzeugen eine Matrix mit $2N$ Zeilen und Spalten:

1	0	0	0	0	0	0	0	2	18	14	6	4	4	9
0	1	0	0	0	0	0	0	9	2	18	14	6	4	4
0	0	1	0	0	0	0	0	4	9	2	18	14	6	4
0	0	0	1	0	0	0	0	4	4	9	2	18	14	6
0	0	0	0	1	0	0	0	6	4	4	9	2	18	14
0	0	0	0	0	1	0	0	14	6	4	4	9	2	18
0	0	0	0	0	0	1	0	18	14	6	4	4	9	2
0	0	0	0	0	0	0	0	19	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	19	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	19	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	19	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	19	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	19

Wir haben die vier Quadranten abgegrenzt, mit der Identität oben links, der Identität mal q entlang der Diagonale unten rechts, einer Nullmatrix unten links und der Zirkulanten der h -Koeffizienten oben rechts. Diese Matrix definiert ein Gitter, das wir als NTRU-Gitter bezeichnen werden. Die Schwierigkeit, ein gitterbasiertes Kryptosystem anzugreifen, hängt von der Schwierigkeit ab, kurze Vektoren im Gitter zu finden.

Die modernsten Algorithmen zum Finden von kurzen Vektoren in Gittern sind Varianten des Lenstra-Lenstra-Lovász (LLL) Algorithmus [8]. Mit den LLL-Algorithmen von Sage Math [9] finden wir kurze Vektoren im Gitter. Die Verwendung des Standard-LLL-Algorithmus erzeugt beispielsweise das Folgende, wobei wir jeder Zeile die Länge des folgenden Vektors vorangestellt haben.

7	0	0	-1	1	0	0	-1	1	0	1	-1	0	-1	0
7	-1	0	0	-1	1	0	0	0	1	0	1	-1	0	-1
7	1	0	0	-1	0	0	-1	-1	0	-1	0	1	0	1
7	-1	1	0	0	-1	0	0	1	-1	0	-1	0	1	0
7	1	1	1	1	1	1	1	0	0	0	0	0	0	0
7	0	1	0	0	1	-1	0	1	0	-1	0	-1	1	0
7	0	0	-1	0	0	-1	1	0	-1	0	1	0	1	-1

95	1	4	-2	-3	-3	-1	3	2	1	0	1	0	-6	2
95	-3	-3	-1	3	1	4	-2	1	0	-6	2	2	1	0
104	0	-1	1	1	3	0	-3	1	0	6	2	1	5	4
95	-3	-1	-4	2	3	3	1	-2	-2	-1	0	-1	0	6
104	0	3	0	1	-1	-1	-3	-5	-4	-1	0	-6	-2	-1
96	-3	-1	4	1	4	-1	-4	0	-5	2	1	1	-1	2
104	-1	1	1	3	0	-3	0	0	6	2	1	5	4	1

Wenn wir die linke und rechte Hälfte der vierten Zeile teilen, erhalten wir

$$-f = [-1, 1, 0, 0, -1, 0, 0]$$

$$-g = [1, -1, 0, -1, 0, 1, 0]$$

Dies sind genau die Negativen der Koeffizientensequenzen für f und g . Wenn wir uns alle Zeilen der Länge 7 außer der fünften genauer ansehen, stellen wir fest, dass sie plus oder minus zirkuläre Verschiebungen voneinander sind, und eine zirkuläre Verschiebung der Koeffizienten der Polynome entspricht der Sternmultiplikation durch eine Potenz von x .

$$f = 1 - x + x^4 = [1, -1, 0, 0, 1, 0, 0]$$

$$g = -1 - x + x^3 - x^5 = [-1, 1, 0, 1, 0, -1, 0]$$

15.7 Die Mathematik des Gitterreduktionsangriffs

Es gibt, das geben wir zu, ein bisschen ein Huhn-und-Ei-Problem in Bezug auf Gitterreduktion, Gitterkryptographie und insbesondere NTRU. Also gehen wir zurück zur Theorie der Gitter.

Lassen Sie

$$h(x) = h_0 + h_1x + \dots + h_{N-1}x^{N-1}$$

ein öffentlicher Schlüssel für eine Instanz von NTRU sein.

Das mit diesem öffentlichen Schlüssel verbundene Gitter L ist das $2N$ -dimensionale Gitter, das von den Zeilen der Matrix M aufgespannt wird, die durch

$$M = \left(\begin{array}{cccc|cccc} 1 & 0 & \dots & 0 & h_0 & h_1 & \dots & h_{N-1} \\ 0 & 1 & \dots & 0 & h_{N-1} & h_0 & \dots & h_{N-2} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & h_1 & h_2 & \dots & h_0 \\ \hline 0 & 0 & \dots & 0 & q & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & q & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & q \end{array} \right)$$

gegeben ist. Wir können die Zeilen dieser Matrix als Koeffizienten für zwei Polynome betrachten

$$(a, b) = ((a_0, a_1, \dots, a_{N-1})(b_0, b_1, \dots, b_{N-1}))$$

Wenn wir annehmen, dass der private Schlüssel $h(x)$ mit zwei Polynomen $f(x)$ und $g(x)$ erstellt wurde, dann können wir einen Angriff auf die NTRU-Instanz in ein Gitterproblem umwandeln mit der folgenden Aussage.

Aussage 15.1 Angenommen, dass $f(x) \circledast h(x) \equiv g(x) \pmod{q}$, dann sei $u(x)$ das Polynom, das

$$f(x) \circledast h(x) = g(x) + qu(x)$$

erfüllt. Mit dieser Definition haben wir, dass

$$(f, -u)M = (f, g)$$

und somit ist der Vektor (f, g) im Gitter L .

Die Schlüsselerkenntnis hier ist, dass die privaten Schlüssel (f, g) im Gitter L sind, aber sie sind auch (per Definition) Polynome mit nur wenigen Nichtnull-Koeffizienten, und diese Koeffizienten sind nur 1 im absoluten Wert. Der Vektor (f, g) ist daher sehr wahrscheinlich ein sehr kurzer Vektor, wenn nicht der kürzeste Vektor im Gitter. Man kann die Wahrscheinlichkeit, dass dies der kürzeste Vektor ist, beweisen, und sie hängt von den Wahlmöglichkeiten der Parameter für die Instanz von NTRU ab.

15.7.1 Andere Angriffe auf NTRU

Der gerade beschriebene Angriff verwendet die Matrix M , um ein Gitter zu erstellen, in dem die privaten Schlüssel f und g wahrscheinlich auftauchen, wenn das Gitter reduziert wird. Ein sehr ähnlicher Angriff kann gegen den Geheimtext gestartet werden. Wenn wir anstelle von

$$M = \begin{pmatrix} I & H \\ 0 & qI \end{pmatrix}$$

wo H die zirkulante Matrix ist, die Matrix

$$\begin{pmatrix} I & H \\ 0 & qI \\ 0 & e \end{pmatrix}$$

verwenden, könnte eine ähnliche Gitterreduktion sehr wohl als einen der kurzen Vektoren den Vektor (m, r) erzeugen.

15.7.2 Gitterreduktion

Also ..., wie sieht Gitterreduktion aus?

Im Fall von zwei Dimensionen sieht die Reduktion von Gittern sehr nach einem zwei-dimensionalen ggT-Algorithmus aus, und der Algorithmus geht auf Gauss zurück.

Gegeben zwei Vektoren

$$\mathbf{v}_1 = [v_{11}, v_{12}]$$

$$\mathbf{v}_2 = [v_{21}, v_{22}]$$

die ein Gitter definieren, können wir die Vektorbasis reduzieren, um den kürzesten Vektor im Gitter mit dem Algorithmus von Algorithmus 1 zu erzeugen.

Algorithm 1 Gaussian reduction in two dimensions

```

1: while True do

2:   if length(v2) ≤ length(v1) then

3:     Swap v1 and v2

4:   end if

5:   Compute  $m = \text{round}(\mathbf{v}_1 \cdot \mathbf{v}_2 / \text{length}(\mathbf{v}_1)^2)$ 

6:   if  $m \neq 0$  then

7:     Return v1 and v2

8:   end if

9:   v2 ← v2 − mv1

10: end while
  
```

Gitterreduktion in höheren Dimensionen ist nicht so einfach wie die Gauss-Reduktion in zwei Dimensionen. Das Problem, die Länge des kürzesten Vektors in einem Gitter zu beweisen, das Problem, die Länge eines kurzen Vektors in einem Gitter zu beweisen, und Algorithmen zur Suche solcher Vektoren wurden seit mehr als zwei Jahrhunderten untersucht. Ein Ergebnis, das für das Studium von NTRU nützlich sein kann, ist die *Gaußsche Heuristik*: Ein zufällig gewähltes Gitter, das durch eine $n \times n$ Matrix L definiert wird, wird einen von Null verschiedenen Vektor \mathbf{v} haben, so dass

$$\|\mathbf{v}\| \approx \frac{n}{2\pi e} |\det L|^{1/n}$$

Der Stand der Technik für Gitterreduktion ist der Lenstra-Lenstra-Lovász-Algorithmus [8], normalerweise einfach als „LLL“ bezeichnet, mit Pseudocode wie in Algorithmus 2.

In diesem Algorithmus sind die Werte $\mu_{k,j}$ und die Zwischenbasisvektoren \mathbf{v}_i^* die Werte aus dem Gram-Schmidt-Orthogonalisierungsprozess von Algorithmus 3. Der Pseudocode für die Algorithmen 2 und 3 stammt von den Algorithmen, die in [6] gezeigt werden.

Algorithm 2 LLL lattice reduction

```

1: Input a basis  $(\mathbf{v}_1, \dots, \mathbf{v}_n)$  for a lattice  $L$ 
2:  $k \leftarrow 2$ 
3:  $\mathbf{v}_1^* \leftarrow \mathbf{v}_1$ 
4: while  $k \leq n$  do

5:   for  $j = 1, 2, \dots, k - 1$  do

6:      $\mathbf{v}_k \leftarrow \mathbf{v}_k - \text{round}(\mu_{k,j})\mathbf{v}_j^*$  {Size reduction}

7:   end for
8:   (Now apply Lovász condition)
9:   if  $\|\mathbf{v}_k^*\|^2 \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) \|\mathbf{v}_{k-1}^*\|^2$  then

10:     $k \leftarrow k + 1$ 

11:   else

12:     Swap  $\mathbf{v}_{k-1}$  and  $\mathbf{v}_k$ 
13:      $k \leftarrow \max(k - 1, 2)$ 

14:   end if

15: end while
16: return Reduced basis  $(\mathbf{v}_1, \dots, \mathbf{v}_n)$ 

```

Algorithm 3 Gram-Schmidt orthogonalization

```

1: Input a basis  $(\mathbf{v}_1, \dots, \mathbf{v}_n)$  for a lattice  $L$ 
2:  $\mathbf{v}_1^* \leftarrow \mathbf{v}_1$ 
3: for  $i = 2, \dots, n$  do

4:   for  $j = 1, \dots, i - 1$  do

5:      $\mu_{ij} \leftarrow (\mathbf{v}_i \cdot \mathbf{v}_j^*) / \|\mathbf{v}_j^*\|^2$ 

6:   end for
7:    $\mathbf{v}_i^* \leftarrow \mathbf{v}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{v}_j^*$ 

8: end for
9: return Orthogonal basis  $(\mathbf{v}_1^*, \dots, \mathbf{v}_n^*)$ 

```

15.8 NTRU Parameterauswahl

Die Forschung zu NTRU ist nach wie vor sehr aktiv. Vorschläge für Parameterauswahlen für NTRU finden Sie auf der Website von Security Innovation [10] und sind in der untenstehenden Tabelle dargestellt. Eine ausführlichere Analyse finden Sie in [11].

	N	q	p
Mäßige Sicherheit	167	128	3
Standard Sicherheit	251	128	3
Hohe Sicherheit	347	128	3
Höchste Sicherheit	503	256	3

15.9 Übungen

1. SageMath berichtet, dass der (wahrscheinlich) zweitkürzeste Vektor im Gitter des Beispiels 15.1 $(-1, 0, 3)$ der Länge $\sqrt{10}$ ist. Überprüfen Sie, ob dieser kurze Vektor im Gitter liegt, indem Sie die Koeffizienten $a, b, c \in \mathbb{Z}^3$ bestimmen, so dass

$$a(1, 4, 9) + b(2, 7, 2) + c(3, 9, 5) = (-2, -2, 0).$$

2. (Wahrscheinlich ist Programmierung erforderlich.) Machen Sie dasselbe für die 14×14 Matrix des Beispiels in Abschn. 15.6: Überprüfen Sie, ob der kurze Vektor der Zeile 1 der von SageMath zurückgegebenen Matrix eine lineare Kombination der Vektoren ist, die das Gitter definieren.
3. (Wahrscheinlich ist Programmierung erforderlich.) Betrachten Sie die Nachricht xy , bestehend aus zwei Zeichen und 16 Bits Länge. Konvertieren Sie diese mit den ASCII-Codes für x und y auf die offensichtliche Weise in eine Ganzzahl und verschlüsseln und entschlüsseln Sie diese Nachricht dann mit dem NTRU-System des Beispiels 15.4. Tipp: Verwenden Sie die beiden Zeichen in der Nachricht, um eine Ganzzahl für die ASCII-Werte zu kodieren, und schreiben Sie diese Ganzzahl dann Basis 3 (da das kleine Modul 3 ist). Das sollte drei Blöcke von Koeffizienten liefern (Sie müssen den letzten mit einer 0 auffüllen), die Sie verschlüsseln und dann entschlüsseln können.
4. (Wahrscheinlich ist Programmierung erforderlich.) Überprüfen Sie, dass die Parameterauswahl wichtig ist. Führen Sie die Verschlüsselung/Entschlüsselung von Problem 3 durch, aber verkleinern Sie dann die Größe von q , bis die Entschlüsselung nicht mehr eindeutig ist.

Literatur

1. D.A. Buell, D.A. Carlson, Y.-C. Chow, K. Culik, N. Deo, R. Finkel, E.N. Houstis, E.M. Jacobson, Z.M. Kedem, J.S. Kowalik, P.J. Kuekes, J.L. Martin, G.A. Michael, N.S. Ostlund, J. Potter, D.K. Pradhan, M.J. Quinn, G.W. Stewart, Q.F. Stout, L. Watson, J. Webb, *Report of the summer workshop on parallel algorithms and architectures*, Report UMIACS TR-86-1, CS-TR-1625 (Supercomputing Research Center and University of Maryland Institute for Advanced Computer Studies, Lanham MD, 1986)
2. C.H. Bennett, G. Brassard, Quantum cryptography: public key distribution and coin tossing, in *International conference on computers, systems, and signal, processing* (1984), S. 175–179
3. T. Monz, D. Nigg, E. Martinez, M.B.I. P. Schindler, R. Rines, S. Wang, I. Chuang, R. Blatt, Realization of a scalable Shor algorithm, in *Science* (2016), S. 1068–1070
4. P.W. Shor, Algorithms for quantum computation: discrete log and factoring, in *Proceedings 35th annual symposium on foundations of computer science* (1994), S. 124–134
5. J. Hoffstein, J. Pipher, J.H. Silverman, NTRU: a ring-based public key cryptosystem, in *Proceedings of the third international symposium on algorithmic number theory, ANTS-III* (1998), S. 267–288
6. J. Hoffstein, J. Pipher, J.H. Silverman, *An introduction to mathematical cryptography* (Springer, 2010)
7. D. Coppersmith, A. Shamir, Lattice attacks on NTRU, in *EUROCRYPT*, Bd. 1233, Lecture notes in computer science, Hrsg. von W. Fumy (1997), S. 52–61
8. A.K. Lenstra, H.W. Lenstra Jr., L. Lov, Factoring polynomials with rational coefficients. *Mathematische Annalen* **261**, 515–534 (1982)
9. The Sage Developers, *Sagemath, the sage mathematics software system (Version 9.2)* (2020). <https://www.sagemath.org>
10. Security Innovation, *NTRU PKCS Tutorial* (2021). <http://www.securityinnovation.com/security-lab/crypto/155.html>
11. J. Hoffstein, J. Pipher, J.M. Schanck, J.H. Silverman, M. Whyte, Z. Zhang, Choosing parameters for NTRUEncrypt, in *Topics in cryptology – CT-RSA 2017*, Bd. 10159, Lecture notes in computer science, Hrsg. von H. Handschuh (2017)

16.1 Einführung

Kurz nach dem ursprünglichen RSA-Papier [1], stellten Rivest, Adleman und Dertouzos eine Frage [2]: Wäre es möglich, eine Datenbank mit verschlüsselten Informationen (wie Finanz- oder Gesundheitsdaten) zu haben, die an einem externen Ort gespeichert sind, die jedoch Berechnungen an den verschlüsselten Daten ohne deren Entschlüsselung ermöglichen würde? Dies würde beispielsweise eine externe Speicherung und Berechnung an den verschlüsselten Daten an der externen Stelle ermöglichen, ohne dass der Eigentümer oder Betreiber der externen Stelle vertraut werden muss.

Wie ursprünglich vorgeschlagen, hat man Klartext S , Geheimtext S' , und eine Entschlüsselungsfunktion $\phi : S' \rightarrow S$ und Verschlüsselungsfunktion $\phi^{-1} : S \rightarrow S'$.

Angenommen, man hat eine Funktion f , die man auf den Klartext anwenden möchte (ein ursprünglicher Vorschlag war der Durchschnittswert ausstehender Kredite einer Finanzinstitution), bräuchten wir eine entsprechende Funktion f' , die auf den Geheimtext operieren würde. Und angenommen, wir haben zwei Dokumente $d_1, d_2 \in S$, für die wir verschlüsselte Versionen haben $\phi^{-1}(d_1) = d'_1$ und $\phi^{-1}(d_2) = d'_2$, könnten wir nach dem Wert von $f(d_1, d_2) = f'(d'_1, d'_2)$ fragen.

Nun, wenn die Verschlüsselungs-/Entschlüsselungsfunktion ϕ ein Homomorphismus wäre, hätten wir

$$\phi(f'(d'_1, d'_2)) = \phi(f'(\phi^{-1}(d_1), \phi^{-1}(d_2))) = f(d_1, d_2),$$

das heißt, die Entschlüsselung des Outputs der Funktion f' wäre das Ergebnis der Anwendung von f auf den Klartext.

Nun, was würde es bedeuten, wenn die Entschlüsselungsfunktion ϕ ein Homomorphismus wäre? Da Funktionen, die in Berechnungen verwendet werden,

auf arithmetischen Operationen beruhen, könnten wir uns vorstellen, dass der Homomorphismus eine Ring-zu-Ring-Abbildung ist, die die Addition und Multiplikation erhält und somit ein Homomorphismus von Ringen ist. Dies steht im Einklang mit der Vorstellung, dass die Funktion f' , die auf den verschlüsselten Daten ausgeführt werden soll, ein Schaltkreis ist; Schaltkreise berechnen Dinge mit Hilfe von Booleschen Operationen, für die Addition und Multiplikation die offensichtlichen Beispiele sind.

Die Einzelheiten der homomorphen Verschlüsselung erfordern eher mehr Hintergrundwissen als der Rest des Materials in diesem Buch. Das, zusammen mit der Tatsache, dass die Forschung zur homomorphen Verschlüsselung noch sehr aktiv ist, macht es unrealistisch, dass wir Details liefern könnten, die sowohl verständlich wären als auch sehr lange als Stand der Technik überleben würden. Aus diesem Grund werden wir nur einen Überblick über dieses wichtige Thema geben, das sicherlich viele Jahre lang im Vordergrund der Kryptographie stehen wird.

16.2 Etwas Homomorphe Verschlüsselung

Es gibt eine Reihe von Kryptosystemen, die als *etwas homomorph* charakterisiert wurden. Zum Beispiel arbeiten RSA-ähnliche Systeme homomorph in Bezug auf Multiplikationen modulo N : Wenn wir Nachrichten m_1 und m_2 und einen privaten Schlüssel e haben, dann haben wir $m_1^e m_2^e \equiv (m_1 m_2)^e \pmod{N}$; modulo N ist die Verschlüsselung des Produkts eindeutig das Produkt der Verschlüsselungen.

16.3 Vollständig Homomorphe Verschlüsselung

Es gibt viele Kryptosysteme, die etwas homomorph sind, in dem Sinne, dass RSA, zum Beispiel, homomorph in Bezug auf Multiplikation arbeitet. Der Begriff der homomorphen Verschlüsselung wurde 1978 vorgeschlagen, aber es war nicht bis zur Doktorarbeit von Craig Gentry und den darauffolgenden Veröffentlichungen [3–5], dass es möglich war, ein Kryptosystem zu haben, das *vollständig homomorph* in dem Sinne war, dass das Kryptosystem jede boolesche Operation auf dem Chiffretext zulassen und die homomorphen Anforderungen erfüllen würde. Seit Gentrys Dissertation wurden eine Reihe von Systemen vorgeschlagen, die meisten davon mit erheblichem Overhead. Die Notwendigkeit, verschlüsselte Daten zu speichern und extern zum Eigentümer des Klartexts zu berechnen, macht dies zu einem aktiven Forschungsbereich.

Wir werden uns auf ein solches System konzentrieren, das von Brakerski, Gentry und Vaikuntanathan vorgeschlagen wurde [6, 7], das wir als BGV bezeichnen werden.

16.4 Ideale Gitter

Viele jüngste kryptographische Vorschläge haben als Grundlage für die kryptographische Maskierung von Klartext den Kontext eines *idealen* in einem Ring verwendet.

Definition 16.1 Gegeben sei ein Ring R mit Addition $+$ und Multiplikation \times , ein *Ideal* I in R ist eine Menge $I \subseteq R$ so, dass I eine Untergruppe unter Addition der additiven Gruppe in R ist, und I ist abgeschlossen unter Multiplikation durch Elemente von R , das heißt, für jedes $r \in R$ und jedes $i \in I$ haben wir $r \times i \in I$.

Gegeben ein ideales I in einem kommutativen Ring R , können wir den *Quotientenring* R/I definieren. Wir stellen zunächst fest, dass ein Ideal eine Äquivalenzrelation erzeugt: für Elemente r und s in R haben wir $r \equiv s$ wenn und nur wenn $r - s \in I$. Dies ermöglicht es uns, die Elemente des Quotientenrings zu definieren, für jedes Element $r \in R$, $r + I = \{r + i | i \in I\}$. Die Operationen im Quotientenring leiten sich von den Operationen in R ab:

$$(r + I) + (s + I) \equiv (r + s) + I$$

$$(r + I) \times (s + I) \equiv (r \times s) + I$$

wo wir die Notation etwas verschmelzen, indem wir die gleichen $+$ und \times Symbole für sowohl R als auch R/I verwenden.

Beispiel 16.1 Wir haben das kanonische Beispiel von Idealen und Quotientenringen in diesem Text verwendet: gegeben eine beliebige ganze Zahl n , dann ist die Menge aller ganzzahligen Vielfachen von n ein Ideal, das wir als $n\mathbb{Z}$ schreiben können, und der Quotientenring kann als

$$\mathbb{Z}/n\mathbb{Z} = \{k + n\mathbb{Z}\}$$

geschrieben werden und besteht aus einer vollständigen Menge von reduzierten Resten modulo n . Für $n = 4$ besteht das Ideal aus allen ganzen Zahlen $4k$, die Vielfache von r sind, und die Arithmetik im Quotientenring ist Kongruenzarithmetik modulo 4:

$$(r + 4t) + (s + 4u) = (r + s) + 4t + 4u = (r + s) + 4(t + u)$$

$$(r + 4t) \times (s + 4u) = (r \times s) + 4ru + 4st + 16tu = (r \times s) + 4(ru + st + 4tu)$$

Kryptographische Schemata, die auf Idealen basieren, funktionieren im Allgemeinen wie folgt: Gegeben ist eine Klartextnachricht m , ein zufälliges Element $i \in I$ wird ausgewählt und der Geheimtext ist $m + i$ im Ring. Dies ermöglicht homomorphe Addition und Multiplikation. Mit einem sorgfältigen Blick auf die Auswahl der Ringe und Ideale kann man eine Falltür als Entschlüsselungsgeheimnis einbauen, die die Umwandlung eines

beliebigen Ringelements in seine Restklasse modulo I ermöglicht. Ohne den geheimen Schlüssel ist die Umwandlung eines beliebigen idealen Elements in das Element, das der Klartext ist, rechnerisch schwierig.

Ein etwas homomorphes Verschlüsselungsschema ist wie folgt, wie in Dijk et al. [8] und anderswo beschrieben. Wir wählen eine große ungerade Zahl (vielleicht eine Primzahl) p im Bereich $[2^{n-1}, 2^n]$. Die Verschlüsselung eines einzelnen Bits m ist eine Zahl kongruent modulo p zur Parität des Bits m . Zur Verschlüsselung wählen wir q und r zufällig und setzen die Geheimtextzahl c auf $c = pq + 2r + m$. Wenn wir p kennen, können wir c modulo p reduzieren, und das Ergebnis $2r + m$ ist genau dann 0 oder 1, wenn m es ist.

Mit diesem Schema kann gezeigt werden, dass man Polynome niedrigen Grades des Geheimtextes auswerten kann, während man die Arithmetik auf dem Geheimtext selbst zulässt. Für ausreichend kleine q, r , vielleicht $r \approx 2\sqrt{n}$ und $q \approx 2^n$, ist das System sicher [8].

Was wir in diesem eher einfachen Ansatz sehen, ähnelt dem, was wir bei NTRU beobachten: Der Geheimtext kann als kleiner „Fehler“ angesehen werden, der zu einem viel größeren Wert hinzugefügt wird. Bei Ganzzahlen nehmen wir „groß“ im üblichen Sinne. Bei Polynomen über Ringen sind wir mehr besorgt über hohe Dimension und spärliche Polynome mit kleinen Koeffizienten.

Eine ausgefeiltere Version des oben genannten Kryptosystems, das zu einem öffentlichen Schlüssel-Kryptosystem wird, ist wie folgt. Mit geeigneten Randbedingungen wählen wir p und probieren eine Reihe von q_i und r_i aus und behalten diese als unseren privaten Schlüssel. Wir berechnen $x_i = q_i p + 2r_i$ für den öffentlichen Schlüssel. Für jedes Bit m des Klartexts wählen wir zufällig eine Reihe von 0, 1-Bits b_i , berechnen die Ganzzahl $B = \sum_i b_i x_i$ und übertragen das Chiffrebit $c = m + B$. Die Entschlüsselung ist somit die Reduzierung von c modulo p , um eine Ganzzahl zu erzeugen, deren Parität der des Klartextbits m entspricht.

Der herkömmliche Angriff auf dieses System ist das, was als das *approximative ggT-Problem* bezeichnet wird. Nämlich, uns wird eine Reihe von Ganzzahlen x_i gegeben, die (vorausgesetzt, wir haben unsere Grenzen richtig gewählt) zufällige große Vielfache eines gemeinsamen Wertes p sind, leicht gestört durch die Addition von „Fehlern“ r_i . Der naive Angriff wäre, für jedes Paar x_j, x_k , alle möglichen ggTs

$$\gcd(x_j - s, x_k - t)$$

für alle möglichen Paare s, t innerhalb der Grenzen für die r_i auszuprobieren.

Ein besserer Ansatz besteht darin, dies auf ein Gitterproblem zu reduzieren. Gegeben zwei beliebige $x_j = q_j p + r_j, x_k = q_k p + r_k$, stellen wir fest, dass der Unterschied $q_k x_j - q_j x_k = q_k r_j - q_j r_k$ klein ist im Vergleich zu x_i , weil wir die zufällige q_i mit kleinen Werten r_i multiplizieren anstatt mit dem viel größeren p .

Wir können dies in ein Gitterproblem umwandeln, indem wir K doppelt so groß wählen wie den größtmöglichen Wert für r_i .

$$(q_0, \dots, q_t) \begin{pmatrix} K & x_1 & \dots & x_t \\ & -x_0 & \dots & \\ & & -x_0 & \dots \\ & & & \dots & -x_0 \end{pmatrix} = (q_0 K, q_0 x_1 - q_1 x_0, \dots, q_0 x_t - q_t x_0)$$

Und dies wäre ein Vektor mit kleinen Koeffizienten und somit ein kurzer Vektor im Gitter.

Beispiel 16.2 Wir veranschaulichen dies mit einem einfachen Beispiel. Wir setzen $p = 2047$ und die (q_i, r_i) ausgewählt aus $(25, 1)$ bis $(34, 10)$ mit den Multiplikatoren q_i und Additiven r_i , die jedes Mal um 1 erhöht werden. Dies ergibt

$$[x_0, \dots, x_9] = [51176, 53224, 55272, 57320, 59368, 61416, 63464, 65512, 67560, 69608].$$

Wenn wir dann die Matrix bilden

$$\begin{pmatrix} 16 & 53224 & 55272 & 57320 & 59368 & 61416 & 63464 & 65512 & 67560 & 69608 \\ 0 & 53224 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -51176 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -51176 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -51176 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -51176 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -51176 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -51176 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -51176 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -51176 \end{pmatrix}$$

und sie mit dem LLL-Algorithmus von SageMath reduzieren, erhalten wir als erste Zeile der reduzierten Matrix den Vektor

$$[400, 24, 48, 72, 96, 120, 144, 168, 192, 216]$$

und eine Lösung für das Problem der Bestimmung der q_i und r_i .

16.5 Lernen mit Fehlern

Das in [8] vorgeschlagene Kryptosystem ist leicht zu verstehen und wurde sowohl in Bezug auf die Implementierung als auch auf die Analyse seiner Sicherheit ausführlich erläutert. Es wurden andere homomorphe Systeme vorgeschlagen, die Polynome und Gitterprobleme verwenden.

BGV, sowie andere vorgeschlagene Systeme, stützen sich auf das Problem des *Lernens mit Fehlern* (LWE). Das *Ring-Lernen mit Fehlern* (RLWE). Problem ähnelt dem Gitterproblem, das vom NTRU-Kryptosystem gestellt wird. Eine Version dieses Problems kann wie folgt beschrieben werden [6].

Wir beginnen mit einem Grad- N Polynom $f(x)$ in einer Variablen x mit ganzzahligen Koeffizienten. Mit diesem können wir einen Ring R von Polynomen mit ganzzahligen Koeffizienten modulo $f(x)$ bilden, und wir können den Ring R_q der Polynome in R mit ihren Koeffizienten reduziert modulo einer ungeraden Ganzzahl q (die nicht prim sein muss) bilden. Die Ringoperationen Addition und Multiplikation sind die offensichtlichen Operationen der Polynomaddition und -multiplikation gefolgt von der Reduktion modulo $f(x)$ und modulo q , genau wie wir es in den Kap. 6 und 15 gemacht haben. Wie wir es früher getan haben, können wir ein Polynomelement in R mit einem Vektor von Koeffizienten

$$g(x) = g_0 + \dots + g_{N-1}x^{N-1} = [g_0, \dots, g_{N-1}]$$

darstellen, wobei die Koeffizienten Ganzzahlen modulo q sind und wir die Koeffizienten in aufsteigenden Potenzen von x ordnen. Wir lassen $\|g\| = \max(|g_i|)$ der maximale absolute Wert der Koeffizienten von $g(x)$ sein.

Die verschiedenen LWE-Probleme über einem Ring wie R können mehr oder weniger intuitiv wie folgt beschrieben werden. Gegeben sind polynomial viele Polynome g_i die gleichmäßig zufällig aus R gewählt werden, gegeben ist ein zufälliges Ringelement s , und gegeben ist eine zufällig gewählte Menge von Elementen e_i mit kleinen Koeffizienten, ist es rechnerisch machbar, die Menge der Paare $(a_i, a_i \cdot s + e_i)$ von einer zufälligen Menge von Paaren aus $R \times R$ zu unterscheiden? Aus intuitiver Sicht handelt es sich hierbei um ein kryptographisches Problem: Wie viele Paare werden benötigt, bevor wir s bestimmen können, indem wir das Rauschen entfernen, das durch das Hinzufügen der e_i ? Ohne die e_i hätten wir ein Problem in der linearen Algebra. Mit der e_i haben wir ein Problem, das auf das Problem der Suche nach kurzen Vektoren in Gittern reduziert wird und daher (soweit heute bekannt) gegen Angriffe mit Quantencomputern resistent ist.

Wir bauen ein symmetrisches Kryptosystem mit geheimem Schlüssel wie folgt auf. Wir beginnen mit einem geheimen Schlüsselpolynom s , das aus R_q ausgewählt wird. Wir ziehen dann eine Probe aus R_q , um ein Polynom a und ein Rauschpolynom e zu erhalten. Das Chiffrebit, das die Verschlüsselung eines Nachrichtenbits m ist, ist das Paar

$$(a, a \cdot s + 2e + m)$$

berechnet in R_q . Die Entschlüsselung erfolgt durch Verwendung von a und dem geheimen Schlüssel s , um das Produkt $a \cdot s$ im Ring R zu berechnen, das zur Extraktion von $2e + m$ aus dem zweiten Element des Paares verwendet wird, und dann modulo 2 reduziert wird, um das Bit m wiederherzustellen. Der Wert q ist ungerade, so dass 2 umkehrbar ist, und vorausgesetzt, q ist groß genug gewählt, und das gezogene e hat

hinreichend kleine Koeffizienten, ist die Extraktion modulo q von $a \cdot s + 2e + m$ zur Erlangung von $2e + m$ wie bei NTRU keine Kongruenz, sondern eine tatsächliche Gleichheit.

16.6 Sicherheit und homomorphe Auswertung von Funktionen

Wir haben bisher weder die Parameter und ihre Auswirkungen auf die Sicherheit kommentiert, noch die Funktionen, die auf den Chiffretexten ausgewertet werden können. Details zu beiden sind leider außerhalb des Rahmens dieses Textes, aber wir können etwas durch Analogie argumentieren. Bei NTRU hatten wir einen großen Integer-Parameter q und einen kleinen Integer-Parameter p und den Grad N des Polynoms $f(x) = x^N - 1$, der den Ring definierte, in dem Verschlüsselung und Entschlüsselung stattfanden. Die Reduktion modulo $f(x)$ und modulo q sollte nicht nur Polynome in Kongruenzklassen erzeugen, sondern die tatsächlichen Polynome mit Koeffizienten modulo p , weil p viel kleiner als q war. Und die Dimension N des Gitters wurde angenommen, groß genug zu sein, dass Gitterreduktionsangriffe rechnerisch unpraktikabel wären.

Die meisten der gleichen Heuristiken gelten hier für homomorphe Verschlüsselung, mit dem Unterschied, dass der Nachweis der Sicherheit für die in diesem Kapitel vorgeschlagenen Methoden durchgeführt werden kann. Der Grad N wird normalerweise als Potenz von 2 genommen und muss groß genug sein, um Gitterreduktionen schwer berechenbar zu machen.

Wichtiger ist es, zu überlegen, welche Funktionen f auf den Chiffretexten in einer sinnvollen Weise berechnen könnten. Mit „berechnen“ meinen wir die Anwendung von Addition und Multiplikation auf Chiffretext. Offensichtlich verbreiten diese Operationen den „Fehler“, und daher wird die willkürliche Berechnung von Polynomen auf Chiffretext nicht zu einer Ausgabe führen, die eindeutig entschlüsselt werden kann. Einer der Hauptbeiträge von Gentry war die Einführung eines rekursiven Verfahrens, mit dem Polynome hohen Grades auf Chiffretext angewendet werden konnten, und eines der Hauptforschungsgebiete ist die Vereinfachung dieses Prozesses [3–5, 7].

16.7 Eine entschuldigende Zusammenfassung

Auf einer Ebene entschuldigen wir uns für den Mangel an Detail in diesem Kapitel. Andererseits scheint die homomorphe Verschlüsselung noch nicht eine geklärte Angelegenheit zu sein, und daher scheint es unrealistisch, zu viel Material zu präsentieren, das bald veraltet sein könnte. Der interessierte Leser sollte planen, auf dem Laufenden zu bleiben mit der Forschungsliteratur, weil der Stand der Technik sich ständig zu ändern scheint.

Literatur

1. R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 120–126 (1978)
2. R.L. Rivest, L. Adleman, M. Dertouzos, On data banks and privacy homo-morphisms. *Foundations of Secure Computation* 169–180 (1978)
3. C. Gentry, A fully homomorphic encryption scheme, Ph.D. thesis (Stanford University, 2009)
4. C. Gentry, Fully homomorphic encryption using ideal lattices, in *Symposium on the Theory of Computing (STOC '09)*, Hrsg. von M. Mitzenmacher (2009), S. 169–178
5. C. Gentry, Toward basing fully homomorphic encryption on worst-case hardness, in *Advances in cryptology – CRYPTO 2010*, Bd. 6223, *Lecture notes in computer science*, Hrsg. von T. Rabin (2010), S. 116–137
6. Z. Brakerski, V. Vaikuntanathan, Efficient fully homomorphic encryption from (standard) LWE, in *52nd Annual Symposium on Foundations of Computer Science* (2011), S. 97–106
7. Z. Brakerski, V. Vaikuntanathan, C. Gentry, Fully homomorphic encryption without bootstrapping, in *Innovations in theoretical computer science* (2012)
8. M. van Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, Fully homomorphic encryption over the integers, in *Advances in cryptology – EUROCRYPT 2010*, Hrsg. von H. Gilbert (2010), S. 24–43

Ein tatsächlicher Erster Weltkrieg Chiffre

A.1 Einleitung

In Kap. VII von Herbert Yardleys *Die amerikanische schwarze Kammer* [1], wird die Methode zur Entschlüsselung einer deutschen diplomatischen Nachricht vorgestellt, und eine zweite Nachricht, die angeblich mit demselben System erstellt wurde, datiert auf den 10. Januar 1919, wird ohne den Entschlüsselungsprozess gegeben, obwohl die resultierende Nachricht in englischer Übersetzung gegeben wird. In diesem Papier werden wir die Entschlüsselung dieser Nachricht nachvollziehen.

A.2 Die Nachricht

Die Nachricht selbst wird auf den Seiten 150–151 von Yardleys Buch gegeben und hier als Abb. A.1 dargestellt.

A.3 Sprachbestimmung

Einerseits sind wir ziemlich sicher, dass dies eine deutsche Nachricht aus Yardleys Kontext ist und dass es sich um eine Transposition und nicht um eine Substitutionschiffre handelt. Andererseits schadet es nie, eine Bestätigung der eigenen Annahmen zu haben, daher führen wir eine Häufigkeitszählung der Buchstaben durch, die als Abb. A.2 erscheint. Wir stellen fest, dass die Nachricht insgesamt 1367 Zeichen enthält.

Diese Häufigkeiten passen ziemlich gut zu einem Modell des Deutschen [2], daher setzen wir unsere Annahme fort, dass es sich um eine Transposition und nicht um eine Substitution handelt.

nogaaaime saeesntraa seienewwei heuamaoeid zcdkeftedt edgeigunri
eceutnninb mhbebanais iteaarukss tdscmoorob aeuoermotd hzzzdigtbt
fceuimlreri eeoemffcea iqeirenuief drisrrbnle enznuhbtprf kgtineenel
anvescalrr adngdceoeu tiailuioerl bkrnnoeeqe hhananvsdf niemineiee
eetreegdmp eilsbihlnu hodciageef sttheetdbe ugmuaudnuu dnsfnenenn
umtralgtu rehnemenbe mntngefsae kltzedrkii rhficnvaks onbtguhewn
thitzmsrmd lghireiscs enpneiette nhvdmvnbvn nrsnecnemn ngepniceuh
eortsgesie eneonfiend wnpkcevemd isrhwlften amucnosazr ahelnehiln
crseamilnb eutceszrth rsaeoszclx mneouhslcu nmenenefae eckerglnra
bgfireubli roznnsseuz csthpusica ufohunbndn betfmmcirt unfnrnsrbna
dsukouiust bmgdreninu lusneadash scecfaoenen ehsmnrgoot erzruierne
incneinfee etkstnbika zeugdednkr ibhideeree aeuneinzet dendaoerea
ighueuoanu uzasruoddi eeemcutiee teanchchdd igrrrrrnso esiereerde
emiehdeade nhdthmnosm elolmeennd rhktendend uockehaete eresfjhok
fhhmkttemn ledsetuehl enimliaern ehzeuesesg snmeuhaimd rrensshikh
rahdhennjh osesedfhn meerneaseh udzsgifmri uoisoehsna deitfeebse
ekamhceant eaoabeunou flrnneizua nfpbhmfnon gusdiport hfrsmndnrl
tmaurrrwini ulnezsknts hdsrdbbnip osedlsuctb ctidafsaue ttunwirhbr
ngnedumiis veurakklne enrctmdtea nsinleimgr iehnlemnlg gkhegdatee
eaaegtero arusrelari graenuinbi eeikdnspni ribhhpkuze tkrseshdne
haravntsee ipreicseuu emozusmudh ipitnndark nalccssgle ursttrlecp
irbdnsaend recoeteian mdttnheamt ntzeomtier nukwmttcke ucebdihtnf
eswgowgeen notzreasnu caahnbgeil ceernsnrta lgghcue

Abb. A.1 Die ursprüngliche Nachricht

Abb. A.2 Buchstabenhäufigkeiten
in der Nachricht

a	79	g	38	m	51	s	81	y	0
b	35	h	65	n	148	t	73	z	24
c	44	i	92	o	46	u	75		
d	65	j	2	p	14	v	9		
e	224	k	29	q	2	w	10		
f	31	l	39	r	90	x	1		

A.4 Eine erste Blockierung

Die ursprüngliche Nachricht scheint in keiner Sprache ein vernünftiger Text zu sein. Nach Yardleys Beispielentschlüsselung gehen wir davon aus, dass alle Buchstaben in der Nachricht um eine feste Anzahl von Positionen von ihrem ursprünglichen Ort verschoben wurden. Um die Buchstaben wieder in ihre korrekten relativen Positionen zu bringen, berücksichtigen wir, dass im Deutschen der Buchstabe ‚c‘ immer von ‚h‘ oder ‚k‘ gefolgt wird. Daher berechnen wir die Abstände (modulo 1367, natürlich) zwischen den 44 Instanzen des Buchstabens ‚c‘ und den 65 Instanzen von ‚h‘. Wir schreiben ein Programm, das diese Unterschiede erzeugt und geben die Ausgabe an einen praktischen Unix-Hack

`Abchdiffs <message | sort -n | uniq -c | sort -n | tail >diffs`
 dessen Ausgabe eindeutig ist. Die häufigsten Buchstabenpositionsunterschiede zwischen
 ‚c‘ und ‚h‘ sind

5	−744
5	778
6	−111
6	124
6	−181
6	−273
6	331
6	446
6	71
26	378

Es ist offensichtlich, dass wir die Nachricht mit Buchstaben blockieren sollten, die einen Unterschied von 378 voneinander haben. Wir tun dies mit einem Programm, das die Nachrichtenfolgennummer und die Zeichentetragramme von Abb. A.3 erzeugt.

A.5 Die Sequenz abschreiben

Im Gegensatz zum ursprünglichen Chiffretext weist diese Sammlung von Tetragrammen (wir werden alle als Tetragramme bezeichnen, auch diejenigen, die zufällig Trigramme sind) deutlich Ähnlichkeiten mit Deutsch auf.

Wenn dieser Code nach demselben System erstellt wurde wie der Code, den Yardley als Beispiel verwendet hat, dann ist unsere nächste Aufgabe, nach einer Sequenz von Sequenznummernunterschieden zu suchen, modulo 378 genommen, und zu versuchen, die Tetragramme mit dieser Sequenz zu verbinden. Wir beginnen, Yardley folgend, mit einer Krippe zur Interpunktion. Der Unix-Hack

```
grep 'k ' tetragrams >cribkomma
grep 'ko ' tetragrams >>cribkomma
grep 'kom ' tetragrams >>cribkomma
grep 'omm ' tetragrams >>cribkomma
grep 'mma ' tetragrams >>cribkomma
grep ' kom' tetragrams >>cribkomma
grep ' omm' tetragrams >>cribkomma
grep ' mma' tetragrams >>cribkomma
grep ' ma' tetragrams >>cribkomma
grep ' a' tetragrams >>cribkomma
```

0	nscg	1	ochd	2	geda	3	andt	4	apie	5	ange	6	iere	7	mira
8	uera	9	etre	10	stre	11	aeng	12	enst	13	ehoe	14	sver	15	ndso
16	tnia	17	rver	18	ahru	19	abes	20	sver	21	enre	22	indl	23	erea
24	nser	25	enmi	26	weig	27	wcer	28	enha	29	iede	30	hmen	31	enau
32	undi	33	agen	34	menb	35	aphi	36	onde	37	eite	38	ichi	39	demk
40	zund	41	chon	42	dess	43	komp	44	eren	45	ftli	46	tsor	47	egli
48	demb	49	tseh	50	eieh	51	denp	52	genk	53	endu	54	ierz	55	gohe
56	unkt	57	nftk	58	rier	59	iens	60	ende	61	cdes	62	ewnh	63	undd
64	tpun	65	nkoe	66	ncch	67	ieka	68	nver	69	beha	70	mmav	71	hden
72	bitt	73	eses	74	bree	75	ahre	76	nwei	77	alsp	78	iffr	79	stje
80	iehi	81	tnoc	82	eaus	83	amke	84	aufu	85	rchu	86	unbe	87	komm
88	ssko	89	satz	90	tztu	91	dres	92	samm	93	chnu	94	meld	95	oleh
96	ondi	97	resp	98	ohei	99	bitt	100	alun	101	enen	102	uchd	103	orla
104	eser	105	renk	106	main	107	omma	108	till	109	dlic	110	hnac	111	zbes
112	zers	113	zung	114	dtel	115	iche	116	gezu	117	bser	118	tzus	119	tret
120	ftst	121	cher	122	ersl	123	usge	124	masc	125	lenp	126	romi	127	eser
128	rzub	129	ichd	130	elan	131	exis	132	omma	133	ende	134	mern	135	ford
136	fuer	137	chne	138	essc	139	also	140	iche	141	quit	142	enke	143	imhi
144	rera	145	enan	146	nehm	147	undd	148	eeht	149	ffen	150	dann	151	renh
152	ieje	153	scha	154	rkom	155	rest	156	bren	157	ngst	158	llez	159	ende
160	erfo	161	nahm	162	zbit	163	ngni	164	ufme	165	hier	166	bren	167	teru
168	punk	169	fbew	170	klam	171	gist	172	tret	173	iohc	174	nzuk	175	ende
176	enzu	177	nssc	178	esge	179	leib	180	aufd	181	nzmi	182	vcrh	183	esit
184	stun	185	chof	186	apie	187	luss	188	rsow	189	rieg	190	acho	191	dasw
192	nung	193	gfae	194	dode	195	chen	196	euin	197	onto	198	enft	199	ubez
200	tder	201	inbe	202	absa	203	ieas	204	lten	205	ufku	206	imac	207	omma
208	rcha	209	lich	210	bren	211	ktab	212	rung	213	nnte	214	nfei	215	oral
216	enoc	217	esae	218	qrbe	219	eber	220	hnun	221	hans	222	adon	223	nsur
224	auft	225	nkla	226	vorl	227	sung	228	ding	229	fueh	230	nsic	231	itzu
232	ebue	233	mma	234	ign	235	ndf	236	erp	237	ieb	238	enh	239	eim
240	enn	241	euf	242	tlo	243	run	244	esg	245	enu	246	ges	247	dad
248	mdi	249	pap	250	eso	251	ihr	252	lst	253	sch	254	bef	255	ich
256	hfr	257	las	258	nom	259	und	260	hen	261	ond	262	der	263	chl
264	ist	265	amm	266	gna	267	eru	268	egr	269	for	270	sow	271	tti
272	ten	273	hri	274	ezu	275	erl	276	tun	277	die	278	bez	279	ers
280	unk	281	gen	282	mit	283	uns	284	ach	285	und	286	der	287	nis
288	und	289	ufb	290	deb	291	nen	292	sei	293	ftp	294	nko	295	ess
296	nre	297	end	298	nbl	299	nis	300	uku	301	mac	302	tzt	303	reb
304	auc	305	lgt	306	gdi	307	ted	308	nda	309	unf	310	rks	311	era
312	hiu	313	nbe	314	eht	315	mit	316	edu	317	nen	318	bew	319	eri
320	mer	321	neh	322	tab	323	ner	324	gun	325	eng	326	fen	327	sie
328	and	329	ezu	330	kem	331	lti	332	tdi	333	zes	334	env	335	dde
336	rau	337	kor	338	iea	339	irk	340	rek	341	hal	342	fin	343	ige
344	che	345	nun	346	ver	347	auc	348	kom	349	sat	350	ond	351	nut
352	bue	353	tza	354	gan	355	uss	356	hri	357	eun	358	wol	359	nde
360	tdi	361	him	362	ieg	363	ter	364	zei	365	mme	366	sch	367	run
368	mtl	369	die	370	lem	371	gen	372	htl	373	ieg	374	rag	375	enk
376	ich	377	che												

Abb. A.3 Nachrichtentetragramme aus der Blockierung bei 378

sammelt aus der Liste der Tetragramme (in der Datei `tetragrams`) alle, die miteinander verbunden werden könnten, um das Wort `komma` („Komma“) zu erzeugen. Wir machen dasselbe für `punkt` („Punkt“) und `klammer` („Klammer“) und berechnen dann von Hand die Unterschiede zwischen den Tetragramm-Sequenznummern.

An diesem Punkt müssen wir anfangen, auf Glück zu hoffen, denn die Buchstabenfolgen sind nicht lang genug, um eindeutige Informationen zu liefern. Für `punkt` erhalten wir beispielsweise die Daten von Abb. A.4, und keiner der Unterschiede wiederholt sich. Für `komma` gibt es jedoch weniger eindeutige Möglichkeiten, und doch wiederholt sich der Unterschied 154. Schließlich haben wir für `klammer`, obwohl wir keine Wiederholungen haben, auch nur drei mögliche Unterschiede von den längeren Krippen. Die Krippen für `komma` und `klammer` sind in Abb. A.5 dargestellt. Wir bemerken auch einige ungewöhnliche (für Deutsch) Tetragramme: `exis` und `quit`. Das erste erfordert fast, dass das nächste Tetragramm mit `t` beginnt, und das zweite erfordert fast, dass das nächste Tetragramm mit `ten` oder `tun` beginnt. Schließlich bemerken wir mehrere Tetragramme, die verwendet werden könnten, um die gebräuchliche Endung `lich` zu bilden. Wenn wir all diese zusammen betrachten, vermuten wir, dass die Abstände 135, 140, 141, 145, 150 und 154 im Kryptogramm vorkommen könnten, weil sie mehr als einmal in unseren Krippen auftreten.

Wir erzeugen daher alle Paare in diesen Abständen und filtern, um diejenigen zu erhalten, die so aussehen, als könnten sie legitimes Deutsch sein. Diese Paare sind in den Abb. A.5, A.6, A.7, A.8, A.9 und A.10 dargestellt.

A.6 Alles zusammenfügen

An diesem Punkt beginnen wir definitiv, uns der Kunst der Kryptanalyse zu nähern, wie sie vor dem Computerzeitalter praktiziert wurde. Wir nehmen an, dass die Tetragramme aus Paaren zu Tripeln, dann zu Quadrupeln usw. zusammengefügt werden sollen. Zu diesem Zweck beginnen wir damit, nach „guten deutschen“ Paaren für eine anfängliche Distanz zu suchen, für die das zweite Tetragramm eines Paares auch das erste Tetragramm eines anderen Paares ist.

Wenn wir mit 135 beginnen, dann könnten eine Reihe von Tripeln von „gutem Deutsch“ mit jedem der anderen Paarsets in den Abb. A.7, A.8, A.9, A.10 und A.11 gebildet werden. Ein Triple jedoch,

282 mit (135) 39 demk (150) 189 rieg

sticht heraus. Im Kontext des Ersten Weltkriegs ist ein Krippenwort, das aussieht wie mit dem `krieg[e]`, verlockend, besonders wenn wir uns die Tetragramme ansehen, die mit `e` beginnen und sehen, dass sowohl Abstände von 140 als auch 145 möglich sind. Wir bemerken auch das mögliche Triple

246 ges (135) 3 andt (150) 153 scha[ft]

Abb. A.4 Gitter für ‚punkt‘

43 komp	16 tnia	307 ted
51 denp	46 tsor	322 tab
77 alsp	49 tseh	332 tdi
97 resp	64 tpun	353 tza
125 lenp	81 tnoc	360 tdi
236 erp	90 tztu	363 ter
249 pap	108 till	
293 ftp	118 tzus	
	119 tret	
64 tpun	167 teru	
	172 tret	
168 punk	200 tder	
	242 tlo	
280 unk	271 tti	
56 unkt	272 ten	
	276 tun	
211 ktab	302 tzt	
64 tpun	→ 211 ktab	$\delta = 147$
43 komp	→ 56 unkt	$\delta = 13$
51 denp	→ 56 unkt	$\delta = 5$
77 alsp	→ 56 unkt	$\delta = 357$
97 resp	→ 56 unkt	$\delta = 337$
125 lenp	→ 56 unkt	$\delta = 309$
236 erp	→ 56 unkt	$\delta = 198$
249 pap	→ 56 unkt	$\delta = 185$
293 ftp	→ 56 unkt	$\delta = 141$
43 komp	→ 280 unk	$\delta = 240$
51 denp	→ 280 unk	$\delta = 229$
77 alsp	→ 280 unk	$\delta = 203$
97 resp	→ 280 unk	$\delta = 183$
125 lenp	→ 280 unk	$\delta = 155$
236 erp	→ 280 unk	$\delta = 44$
249 pap	→ 280 unk	$\delta = 31$
293 ftp	→ 280 unk	$\delta = 365$

Abb. A.5 Gitter für ‚komma‘
und ‚klammer‘

225 nkla	→ 365 mme	$\delta = 140$
170 klam	→ 320 mern	$\delta = 150$
170 klam	→ 134 mern	$\delta = 342$
88 ssko	→ 233 mma	$\delta = 145$
88 ssko	→ 70 mmav	$\delta = 360$
294 nko	→ 233 mma	$\delta = 317$
294 nko	→ 70 mmav	$\delta = 154$
154 rkom	→ 106 main	$\delta = 330$
154 rkom	→ 124 masc	$\delta = 348$
154 rkom	→ 301 mac	$\delta = 157$
348 kom	→ 106 main	$\delta = 136$
348 kom	→ 124 masc	$\delta = 154$
348 kom	→ 301 mac	$\delta = 331$

2 geda 137 chne	91 dres 226 vorl	246 ges 3 andt
12 enst 147 undd	102 uchd 237 ieb	249 pap 6 iere
15 ndso 150 dann	123 usge 258 nom	253 sch 10 stre
24 nser 159 ende	137 chne 272 ten	257 las 14 sver
26 weig 161 nahm	140 iche 275 erl	282 mit 39 demk
28 enha 163 ngni	141 quit 276 tun	285 und 42 dess
30 hmen 165 hier	145 enan 280 unk	286 der 43 komp
33 agen 168 punk	150 dann 285 und	291 nen 48 demb
37 eite 172 tret	156 bren 291 nen	296 nte 53 endu
40 zund 175 ende	187 luss 322 tab	300 uku 57 nftk
41 chon 176 enzu	192 nung 327 sie	315 mit 72 bitt
44 eren 179 leib	207 omma 342 fin	318 bew 75 ahre
48 demb 183 esit	209 lich 344 che	320 mer 77 alsp
51 denp 186 apie	210 bren 345 nun	327 sie 84 aufu
69 beha 204 lten	211 ktab 346 ver	336 rau 93 chnu
75 ahre 210 bren	212 rung 347 auc	354 gan 111 zbes
77 alsp 212 rung	213 nnte 348 kom	364 zei 121 cher
84 aufu 219 eber	219 eber 354 gan	372 htl 129 ichd
86 unbe 221 hans	228 ding 363 ter	373 ieg 130 elan
87 komm 222 adon	242 tlo 377 che	374 rag 131 exis
89 satz 224 auft	245 enu 2 geda	376 ich 133 ende

Abb. A.6 Paare in einem Abstand von 135

2 geda 142 enke	113 zung 253 sch	201 inbe 341 hal
4 apie 144 rera	115 iche 255 ich	207 omma 347 auc
6 iere 146 nehm	127 eser 267 eru	209 lich 349 sat
15 ndso 155 rest	131 exis 271 tti	211 ktab 351 nut
30 hmen 170 klam	135 ford 275 erl	216 enoc 356 hri
37 eite 177 nssc	136 fuer 276 tun	225 nkla 365 mme
43 komp 183 esit	137 chne 277 die	259 und 21 enre
44 eren 184 stun	140 iche 280 unk	275 erl 37 eite
45 ftli 185 chof	150 dann 290 deb	277 die 39 demk
60 ende 200 tder	153 scha 293 ftp	286 der 48 demb
63 undd 203 ieas	155 rest 295 ess	315 mit 77 alsp
90 tztu 230 nsic	156 bren 296 nte	327 sie 89 satz
94 meld 234 ign	159 ende 299 nis	349 sat 111 zbes
99 bitt 239 eim	175 ende 315 mit	351 nut 113 zung
103 orla 243 run	184 stun 324 gun	369 die 131 exis
105 renk 245 enu	188 rsow 328 and	374 rag 136 fuer
107 omma 247 dad	189 rieg 329 ezu	

Abb. A.7 Paare in einem Abstand von 140

und erkennen das deutsche Wort *Gesandtschaft*, oder „legation.“ Noch besser, wir bemerken nur drei Tetragramme (45 ftli, 120 ftst, und 293 ftp), die mit dem notwendigen ft beginnen, und das dritte davon befindet sich in einem Abstand von 140

5 ange 146 nehm	195 chen 336 rau	293 ftp 56 unkt
24 nser 165 hier	200 tder 341 hal	320 mer 83 amke
84 aufu 225 nkla	203 ieas 344 che	347 auc 110 hnac
99 bitt 240 enn	228 ding 369 die	358 wol 121 cher
131 exis 272 ten	265 amm 28 enha	365 mme 128 rzub
136 fuer 277 die	269 for 32 undi	373 ieg 136 fuer
139 also 280 unk	275 erl 38 ichi	376 ich 139 also
140 iche 281 gen	285 und 48 demb	

Abb. A.8 Paare in einem Abstand von 141

29 iede 174 nzuk	134 mern 279 ers	238 enh 5 ange
31 enau 176 enzu	136 fuer 281 gen	270 sow 37 eite
44 eren 189 rieg	137 chne 282 mit	272 ten 39 demk
56 unkt 201 inbe	153 scha 298 nbl	286 der 53 endu
72 bitt 217 esae	160 erfo 305 lgt	304 auc 71 hden
80 iehi 225 nkla	180 aufd 325 eng	320 mer 87 komm
85 rchu 230 nsic	189 rieg 334 env	322 tab 89 satz
88 ssko 233 mma	192 nung 337 kor	342 fin 109 dlic
99 bitt 244 esg	204 lten 349 sat	358 wol 125 lenp
113 zung 258 nom	209 lich 354 gan	360 tdi 127 eser
131 exis 276 tun	213 nnte 358 wol	373 ieg 140 iche
132 omma 277 die	216 enoc 361 him	

Abb. A.9 Paare in einem Abstand von 145

3 andt 153 scha	136 fuer 286 der	265 amm 37 eite
6 iere 156 bren	144 rera 294 nko	273 hri 45 ftli
30 hmen 180 aufd	147 undd 297 end	276 tun 48 demb
39 demk 189 rieg	160 erfo 310 rks	288 und 60 ende
45 ftli 195 chen	165 hier 315 mit	309 unf 81 tnoc
56 unkt 206 imac	170 klam 320 mer	314 eht 86 unbe
57 nftk 207 omma	183 esit 333 zes	315 mit 87 komm
89 satz 239 eim	192 nung 342 fin	317 nen 89 satz
92 samm 242 tlo	201 inbe 351 nut	320 mer 92 samm
101 enen 251 ihr	204 lten 354 gan	327 sie 99 bitt
107 omma 257 las	219 eber 369 die	332 tdi 104 eser
110 hnac 260 hen	224 auf 374 rag	359 nde 131 exis
111 zbes 261 ond	226 vorl 376 ich	363 ter 135 ford
129 ichd 279 ers	233 mma 5 ange	368 mtl 140 iche
132 omma 282 mit	237 ieb 9 etre	369 die 141 quit
134 mern 284 ach	242 tlo 14 sver	
135 ford 285 und	251 ihr 23 erea	

Abb. A.10 Paare in einem Abstand von 150

12 enst 166 bren	150 dann 304 auc	277 die 53 endu
23 erea 177 nssc	157 ngst 311 era	284 ach 60 ende
24 nser 178 esge	159 ende 313 nbe	292 sei 68 nver
30 hmen 184 stun	161 nahm 315 mit	294 nko 70 mmav
37 eite 191 dasw	168 punk 322 tab	296 nte 72 bitt
56 unkt 210 bren	180 aufd 334 env	297 end 73 eses
104 eser 258 nom	186 apie 340 rek	313 nbe 89 satz
111 zbes 265 amm	189 rieg 343 ige	325 eng 101 enen
115 iche 269 for	192 nung 346 ver	328 and 104 eser
125 lenp 279 ers	195 chen 349 sat	341 hal 117 bser
134 mern 288 und	204 lten 358 wol	348 kom 124 masc
136 fuer 290 deb	219 eber 373 ieg	362 ieg 138 essc
137 chne 291 nen	227 sung 3 andt	366 sch 142 enke
140 iche 294 nko	262 der 38 ichi	374 rag 150 dann
144 rera 298 nbl	264 ist 40 zund	

Abb. A.11 Paare in einem Abstand von 154

von (153 scha. Dies bestätigt unsere Feststellung, dass es viele Fälle von Tripeln in Abständen von 150 gefolgt von 140 gibt, die gutes Deutsch zu sein scheinen. Wir wagen eine wilde Vermutung und setzen Quadrupel von Tetragrammen aus den Abständen 135, 150 und 140 zusammen und finden zumindest die in Abb. A.12 gezeigten Fälle

Abb. A.12 Quadrupel in Abständen von 135, 150 und 140

2 geda 137 chne 287 nis 49 tseh
12 enst 147 undd 297 end 59 iens
30 hmen 165 hier 315 mit 77 alsp
48 demb 183 esit 333 zes 95 oleh
66 ncch 201 inbe 351 nut 113 zung
69 beha 204 lten 354 gan 116 gezu
84 aufu 219 eber 369 die 131 exis
89 satz 224 auft 374 rag 136 fuer
102 uchd 237 ieb 9 etre 149 ffen
138 essc 273 hri 45 ftli 185 chof
156 bren 291 nen 63 undd 203 ieas
174 nzuk 309 unf 81 tnoc 221 hans
192 nung 327 sie 99 bitt 239 eim
246 ges 3 andt 153 scha 293 ftp
249 pap 6 iere 156 bren 296 nte
254 bef 11 aeng 161 nahm 301 mac
264 ist 21 enre 171 gist 311 era
282 mit 39 demk 189 rieg 329 ezu
300 uku 57 nftk 207 omma 347 auc
318 bew 75 ahre 225 nkla 365 mme
336 rau 93 chnu 243 run 5 ange
354 gan 111 zbes 261 ond 23 erea
375 enk 132 omma 282 mit 44 eren

von möglichem gutem Deutsch. Wir stellen fest, dass es in dieser Liste ein oder zwei Störungen geben könnte, insbesondere bei den Quadrupeln, die mit den Tetragrammen 48 und 66 beginnen, aber diese Liste scheint recht vielversprechend.

A.7 Weiteres Raten

Nun vertiefen wir uns weiter in der Hoffnung, die richtigen Verbindungen zu erraten. Das Vierfache, das mit Tetragramm 84 beginnt, verlangt nach einem Anfangsbuchstaben *t*, die Vierfachen, die mit den Tetragrammen 254 und 347 beginnen, müssen von Vierfachen mit dem Anfangsbuchstaben *c* gefolgt werden, und das Vierfache, das mit 318 beginnt, scheint ein Tetragramm zu benötigen, das mit *r* beginnt. Wenn wir nach Abständen für diese Tetragramme suchen, erhalten wir wiederholte Möglichkeiten der Abstände 9, 71 und 141. Allerdings ermöglicht uns die 141 auch,

246 ges (135) 3 andt (150) 153 scha (140) 293 ftp

zu

246 ges (135) 3 andt (150) 153 scha (140) 293 ftp (141) unkt

zu vervollständigen und wir entscheiden uns, dies zu versuchen. Die daraus resultierenden möglichen deutschen Zeichenketten sind in Abb. A.13 dargestellt.

Wir sind noch nicht fertig. Wahrscheinlich brauchen wir ein *z*, um dem Tetragramm 272 zu folgen. Wir brauchen ein *h*, um dem Tetragramm 110 zu folgen. Und wir

Abb. A.13 Fünffache in Abständen von 135, 150, 140, 141

12 enst 147 undd 297 end 59 iens 200 tder
 30 hmen 165 hier 315 mit 77 alsp 218 qrbe
 48 demb 183 esit 333 zes 95 oleh 236 erp
 66 ncch 201 inbe 351 nut 113 zung 254 bef
 69 beha 204 lten 354 gan 116 gezu 257 las
 84 aufu 219 eber 369 die 131 exis 272 ten
 89 satz 224 auft 374 rag 136 fuer 277 die
 102 uchd 237 ieb 9 etre 149 ffen 290 deb
 138 essc 273 hri 45 ftli 185 chof 326 fen
 156 bren 291 nen 63 undd 203 ieas 344 che
 174 nzuk 309 unf 81 tnoc 221 hans 362 ieg
 228 ding 363 ter 135 ford 275 erl 38 ichi
 233 mma 368 mtl 140 iche 280 unk 43 komp
 246 ges 3 andt 153 scha 293 ftp 56 unkt
 282 mit 39 demk 189 rieg 329 ezu 92 samm
 300 uku 57 nftk 207 omma 347 auc 110 hnac
 318 bew 75 ahre 225 nkla 365 mme 128 rzub
 336 rau 93 chnu 243 run 5 ange 146 nehm
 354 gan 111 zbes 261 ond 23 erea 164 ufme

könnten vermuten, dass en dem Tetragramm 92 folgen würde. Wenn wir nach doppelten Abständen unter diesen suchen, kommen wir auf 146, 298 und 339 als Möglichkeiten. Die Wahl von 146 scheint jedoch gut zu sein, wenn wir bemerken, dass dies

354 gan (135) 111 zbes (150) 261 ond (140) 23 erea (141) 164 ufme

fortsetzen würde, um

354 gan 111 zbes 261 ond 23 erea 164 ufme 310 rks

zu werden.

A.8 Fortsetzung der Sequenz

Wir sammeln Sextupel von Paaren in Abb. A.14. Einige Dinge werden sehr klar. Das ungerade pqr in der Zeichenkette, die mit 30 beginnt, muss ein Name sein, da der Text sagt, dass „hiermit als“. Tetragramm 95 in Zeile 3 ist ein Durcheinander, ebenso wie Tetragramm 66 in Zeile 4. Wir haben einige der ungeraden Zeilen nach unten verschoben. Zeile 8, die mit Tetragramm 156 beginnt, spricht von „Verbrennung“ und davon, etwas mit der Asche zu tun.

Krippen von hier an sind etwas schwieriger. Die einzigen Zeilen, die vielversprechend aussehen, sind die zweite Zeile und die letzte Zeile über der mittleren Trennlinie. Die

```

12 enst 147 undd 297 end 59 iens 200 tder 346 ver
30 hmen 165 hier 315 mit 77 alsp 218 qrbe 364 zei
48 demb 183 esit 333 zes 95 oleh 236 erp 4 apie
66 ncch 201 inbe 351 nut 113 zung 254 bef 22 indl
69 beha 204 lten 354 gan 116 gezu 257 las 25 enmi
84 aufu 219 eber 369 die 131 exis 272 ten 40 zund
89 satz 224 auft 374 rag 136 fuer 277 die 45 ftli
102 uchd 237 ieb 9 etre 149 ffen 290 deb 58 rier
138 essc 273 hri 45 ftli 185 chof 326 fen 94 meld
156 bren 291 nen 63 undd 203 ieas 344 che 112 zers
174 nzuk 309 unf 81 tnoc 221 hans 362 ieg 130 elan
228 ding 363 ter 135 ford 275 erl 38 ichi 184 stun
233 mma 368 mtl 140 iche 280 unk 43 komp 189 rieg
246 ges 3 andt 153 scha 293 ftp 56 unkt 202 absa
282 mit 39 demk 189 rieg 329 ezu 92 samm 238 enh
300 uku 57 nftk 207 omma 347 auc 110 hnac 256 hfr
318 bew 75 ahre 225 nkla 365 mme 128 rzub 274 ezu
336 rau 93 chnu 243 run 5 ange 146 nehm 292 sei
354 gan 111 zbes 261 ond 23 erea 164 ufme 310 rks

```

Abb. A.14 Sextupel in den Abständen 135, 150, 140, 141, 146

erste könnte mit [ne]hmen beginnen und mit qrbe zei [ch] enden. Aus der letzten Zeile könnten wir eine Notwendigkeit für aufmerksam ableiten. Beide können mit einem Abstand von 151 untergebracht werden, und da es nur zwei Möglichkeiten für aufmerksam gibt, sind wir uns ziemlich sicher, dass dies korrekt ist. Wir präsentieren Septupel in Abb. A.15.

Weitere Krippen und Beobachtungen: Zeile 3, Tetragramm 95, ist wahrscheinlich olch. Zeile 4, Tetragramm 66, ist wahrscheinlich noch. Zeile 4, Tetragramm 173, ist wahrscheinlich ichc.

Zeile 2 sollte wahrscheinlich mit einem n oder einem t gefolgt werden.

Zeile 8: „Brennen, und die Asche ...“ Man konsultiert das Wörterbuch und findet heraus, dass *zerschlagen* das Wort ist, also würden wir nach agen suchen, um dieser Zeile zu folgen. Es gibt nur ein Tetragramm, 33 agen, das passen würde, in einem Abstand von 148.

Zeile 12: Wir könnten nach *zusammenhaengen...* suchen. Tetragramm 159, ende, würde zu einer Entfernung von 148 passen.

Zeile 14: Wir müssen dies fast sicher mit ch fortsetzen. Tetragramm 195, chen, würde zu einer Entfernung von 148 passen.

Wir präsentieren Oktupel in Abb. A.16. Offensichtlich haben viele davon Störungen, was wir erwarten würden. Allerdings würden wir an diesem Punkt (oder etwas früher, wenn wir klüger wären), bemerken, dass die Zeilen am häufigsten um 18 in den Tetragramm-Sequenznummern abweichen. Da $378 = 18 \cdot 21$, vermuten wir, dass die Nachricht in 21 Zeilen blockiert ist, 18 Tetragramme pro Zeile, und wir nehmen in unsere Strings von mutmaßlichem Deutsch alle Zeilen auf, auch wenn sie zu diesem Zeitpunkt nicht unbedingt wie gutes Deutsch aussehen.

```

12 enst 147 undd 297 end 59 iens 200 tder 346 ver 119 tret
30 hmen 165 hier 315 mit 77 alsp 218 qrbe 364 zei 137 chne
48 demb 183 esit 333 zes 95 oleh 236 erp 4 apie 155 rest
66 ncch 201 inbe 351 nut 113 zung 254 bef 22 indl 173 iohc
84 aufu 219 eber 369 die 131 exis 272 ten 40 zund 191 dasw
102 uchd 237 ieb 9 etre 149 ffen 290 deb 58 rier 209 lich
138 essc 273 hri 45 ftli 185 chof 326 fen 94 meld 245 enu
156 bren 291 nen 63 undd 203 ieas 344 che 112 zers 263 chl
174 nzuk 309 unf 81 tnoc 221 hans 362 ieg 130 elan 281 gen
228 ding 363 ter 135 ford 275 erl 38 ichi 184 stun 335 dde
246 ges 3 andt 153 scha 293 ftp 56 unkt 202 absa 353 tza
282 mit 39 demk 189 rieg 329 ezu 92 samm 238 enh 11 aeng
300 uku 57 nftk 207 omma 347 auc 110 hnac 256 hfr 29 iede
318 bew 75 ahre 225 nkla 365 mme 128 rzub 274 ezu 47 egli
336 rau 93 chnu 243 run 5 ange 146 nehm 292 sei 65 nkoe
354 gan 111 zbes 261 ond 23 erea 164 ufme 310 rks 83 amke

```

Abb. A.15 Septupel in den Abständen 135, 150, 140, 141, 146, 151

Abb. A.16 Oktupel in den
Abständen 135, 150, 140, 141,
146, 151, 148

12	147	297	59	200	346	119	267
enst	undd	end	iens	tder	ver	tret	eru
30	165	315	77	218	364	137	285
hmen	hier	mit	alsp	qrbe	zei	chne	und
48	183	333	95	236	4	155	303
demb	esit	zes	oleh	erp	apie	rest	reb
66	201	351	113	254	22	173	321
ncch	inbe	nut	zung	bef	indl	iohc	neh
84	219	369	131	272	40	191	339
aufu	eber	die	exis	ten	zund	dasw	irk
102	237	9	149	290	58	209	357
uchd	ieb	etre	ffen	deb	rier	lich	eun
120	255	27	167	308	76	227	375
ftst	ich	wcer	teru	nda	nwei	sung	enk
138	273	45	185	326	94	245	15
essc	hri	ftli	chof	fen	meld	enu	ndso
156	291	63	203	344	112	263	33
bren	nen	undd	ieas	che	zers	chl	agen
174	309	81	221	362	130	281	51
nzuk	unf	tnoc	hans	ieg	elan	gen	denp
192	327	99	239	2	148	299	69
nung	sie	bitt	eim	geda	eeht	nis	beha
210	345	117	257	20	166	317	87
bren	nun	bser	las	sver	bren	nen	komm
228	363	135	275	38	184	335	105
ding	ter	ford	erl	ichi	stun	dde	renk
246	3	153	293	56	202	353	123
ges	andt	scha	ftp	unk	absa	tza	usge
264	21	171	311	74	220	371	141
ist	enre	gist	era	bree	hnun	gen	quit
282	39	189	329	92	238	11	159
mit	demk	rieg	ezu	samm	enh	aeng	ende
300	57	207	347	110	256	29	177
uku	nftk	omma	auc	hnac	hfr	iede	nssc
318	75	225	365	128	274	47	195
bew	ahre	nkla	mme	rzub	ezu	egli	chen
336	93	243	5	146	292	65	213
rau	chnu	run	ange	nehm	sei	nkoe	nnte
354	111	261	23	164	310	83	231
gan	zbes	ond	erea	ufme	rks	amke	itzu
372	129	279	41	182	328	101	249
htl	ichd	ers	chon	vcrh	and	enen	pap

Wir schauen uns nun unsere Sequenzen an, nachdem wir zuerst alle 168 Tetragramme entfernt haben, die in Abb. A.16 erscheinen.

Wir vermuten, dass Tetragramm 12 möglicherweise von einem d vorangestellt wird. Für dies erhalten wir Abstände von 30, 58, 84, 142, 294 und 358.

Wir vermuten, dass das Tetragramm 102 möglicherweise von einem a vorangestellt wird. Dafür erhalten wir Abstände von 35, 74, 86, 94, 95, 142, 214, 247, 272, 336, 348, 373 und 377.

Wir vermuten, dass das Tetragramm 300 möglicherweise von einem z vorangestellt wird. Dafür erhalten wir Abstände von 22, 101, 142, 211 und 246.

Der gemeinsame Wert hier ist 142, und in Abb. A.17 präsentieren wir Sequenzen der Länge neun.

Von den verbleibenden Tetragrammen wäre die einzige gute Wahl, um auf Tetragramm 141, *quit*, zu folgen, Tetragramm 276, *tun* in einer Entfernung von 135. Dies ergibt Abb. A.18.

Jetzt zu etwas mehr Spickerei.

Am Ende der Zeile 1 könnten wir ein t erwarten.

Am Ende der Zeile 2 könnten wir ein r erwarten.

Am Ende der Zeile 7 könnten wir ein g erwarten.

Am Ende der Zeile 8 könnten wir ein mma erwarten.

Am Ende der Zeile 11 könnten wir ein ma erwarten.

Diese beiden letzteren liefern einige Hinweise, da es nur zwei Tetragramme für Zeile 8 gibt, nämlich 70, *mmav* und 233, *mma*, in den Abständen 154 und 84.

Darüber hinaus gibt es nur zwei Tetragramme für Zeile 11, nämlich 106, *main*, 124, *masc*, und 301, *mac*, in den Abständen 136, 154 und 331.

Wir gehen mit der üblichen 154, und stellen fest, dass wir die erwarteten Buchstaben für die Zeilen 1, 2 und 7 erhalten.

Nun, in der letzten Zeile, müssen wir ein Tetragramm finden, das mit h beginnt, und es gibt nur noch drei übrig: 71, *hden*, 260, *hen*, und 341, *hal*, in den Abständen 145, 334, und 37. Alle drei sind zulässig, keines würde uns zwingen, ein Tetragramm zu verwenden, das wir bereits verwendet haben, aber 145 erzeugt eindeutig überlegenes Deutsch.

Mit dem Verb am Ende der Zeile 3 vermuten wir, dass wir mit *unk* folgen müssen, was jetzt nur noch Tetragramm 280 in einer Entfernung von 155 wäre.

Das funktioniert.

Zeile 12 erfordert nun, dass das Wort *punkt* vervollständigt wird, für eine Entfernung von 147 bis zum Tetragramm 211.

Der Pfad scheint an dieser Stelle zu stocken, also arbeiten wir am anderen Ende der Fäden. Zeile 9 benötigt einen Vokal vor dem Tetragramm 158. Wir versuchen es mit dem *uera*, da es das einzige verbleibende a ist, und stellen fest, dass dies auch unsere lange verzögerten *klam* und *mer* zusammenbringt.

Wir haben jetzt *oheimdienst* in Zeile 5. Sicherlich ist das führende o ein Fehler, und es soll *geheimdienst* sein. Nur 0 *nscg*, das auch ein Fehler sein muss, und 325 *eng* würden dafür funktionieren. Die Unterschiede betragen 98 und 151. Wir versuchen beide und die 151 erzeugt konsistenteres Deutsch.

An dieser Stelle haben wir nur noch eine Spalte übrig. Brute Force auf einem gleitenden Streifen von Tetragrammen, oder alternativ die Vermutung, dass das Tetragramm 0 bis 145 mit einem Abstand von 145 dem Muster des Kryptogramms entspricht, liefert die endgültige Nachricht.

Abb. A.17 Sequenzen
der Länge neun, Abstände
142,135,150,140,141,146,
151,148

248	12	147	297	59	200	346	119	267
mdi	enst	undd	end	iens	tder	ver	tret	eru
266	30	165	315	77	218	364	137	285
gna	hmen	hier	mit	alsp	qrbe	zei	chne	und
284	48	183	333	95	236	4	155	303
ach	demb	esit	zes	oleh	erp	apie	rest	reb
302	66	201	351	113	254	22	173	321
tzst	ncch	inbe	nut	zung	bef	indl	iohc	neh
320	84	219	369	131	272	40	191	339
mer	aufu	eber	die	exis	ten	zund	dasw	irk
338	102	237	9	149	290	58	209	357
iea	uchd	ieb	etre	ffen	deb	rier	lich	eun
356	120	255	27	167	308	76	227	375
hri	ftst	ich	wcer	teru	nda	nwei	sung	enk
374	138	273	45	185	326	94	245	15
rag	essc	hri	ftli	chof	fen	meld	enu	ndso
14	156	291	63	203	344	112	263	33
sver	bren	nen	undd	ieas	che	zers	chl	agen
32	174	309	81	221	362	130	281	51
undi	nzuk	unf	tnoc	hans	ieg	elan	gen	denp
50	192	327	99	239	2	148	299	69
eieh	nung	sie	bitt	eim	geda	eeht	nis	beha
68	210	345	117	257	20	166	317	87
nver	bren	nun	bser	las	sver	bren	nen	komm
86	228	363	135	275	38	184	335	105
unbe	ding	ter	ford	erl	ichi	stun	dde	renk
104	246	3	153	293	56	202	353	123
eser	ges	andt	scha	ftp	unk	absa	tza	usge
122	264	21	171	311	74	220	371	141
ersl	ist	enre	gist	era	bree	hnun	gen	quit
140	282	39	189	329	92	238	11	159
iche	mit	demk	rieg	ezu	samm	enh	aeng	ende
158	300	57	207	347	110	256	29	177
llez	uku	nftk	omma	auc	hnac	hfr	iede	nssc
176	318	75	225	365	128	274	47	195
enzu	bew	ahre	nkla	mme	rzub	ezu	egli	chen
194	336	93	243	5	146	292	65	213
dode	rau	chnu	run	ange	nehm	sei	nkoe	nnte
212	354	111	261	23	164	310	83	231
rung	gan	zbes	ond	erea	ufme	rks	amke	itzu
230	372	129	279	41	182	328	101	249
nsic	htl	ichd	ers	chon	vcrh	and	enen	pap

A.9 Zusammenstellung der endgültigen Nachricht

Wir präsentieren unten den Text der Nachricht. Unsere Sequenz von Unterschieden ist
145, 151, 150, 142, 135, 150, 140, 141, 146, 151, 148, 135, 154, 145, 155, 147, 138

Abb. A.18 Länge zehn
Sequenzen, Abstände 142, 135,
150, 140, 141, 146, 151, 148,
145

14	156	291	63	203	344	112	263	33	168
sver	bren	nen	undd	ieas	che	zers	chl	agen	punk
32	174	309	81	221	362	130	281	51	186
undi	nzuk	unf	tnoc	hans	ieg	elan	gen	denp	apie
50	192	327	99	239	2	148	299	69	204
eieh	nung	sie	bitt	eim	geda	eeht	nis	beha	lten
68	210	345	117	257	20	166	317	87	222
nver	bren	nun	bser	las	sver	bren	nen	komm	adon
86	228	363	135	275	38	184	335	105	240
unbe	ding	ter	ford	erl	ichi	stun	dde	renk	enn
104	246	3	153	293	56	202	353	123	258
eser	ges	andt	scha	ftp	unkt	absa	tza	usge	nom
122	264	21	171	311	74	220	371	141	276
ersl	ist	enre	gist	era	bree	hnun	gen	quit	tun
140	282	39	189	329	92	238	11	159	294
iche	mit	demk	rieg	ezu	samm	enh	aeng	ende	nko
158	300	57	207	347	110	256	29	177	312
llez	uku	nftk	omma	auc	hnac	hfr	iede	nssc	hiu
176	318	75	225	365	128	274	47	195	330
enzu	bew	ahre	nkla	mme	rzub	ezu	egli	chen	kem
194	336	93	243	5	146	292	65	213	348
dode	rau	chnu	run	ange	nehm	sei	nkoe	nnte	kom
212	354	111	261	23	164	310	83	231	366
rung	gan	zbes	ond	erea	ufme	rks	amke	itzu	sch
230	372	129	279	41	182	328	101	249	6
nsic	htl	ichd	ers	chon	vcrh	and	enen	pap	iere
248	12	147	297	59	200	346	119	267	24
mdi	enst	undd	end	iens	tder	ver	tret	eru	nser
266	30	165	315	77	218	364	137	285	42
gna	hmen	hier	mit	alsp	qrbe	zei	chne	und	dess
284	48	183	333	95	236	4	155	303	60
ach	demb	esit	zes	oleh	erp	apie	rest	reb	ende
302	66	201	351	113	254	22	173	321	78
tzt	ncch	inbe	nut	zung	bef	indl	iohc	neh	iffr
320	84	219	369	131	272	40	191	339	96
mer	aufu	eber	die	exis	ten	zund	dasw	wirk	ondi
338	102	237	9	149	290	58	209	357	114
iea	uchd	ieb	etre	ffen	deb	rier	lich	eun	dtel
356	120	255	27	167	308	76	227	375	132
hri	ftst	ich	wcer	teru	nda	nwei	sung	enk	omma
374	138	273	45	185	326	94	245	15	150
rag	essc	hri	ftli	chof	fen	meld	enu	ndso	dann

die wir in den Abb. A.19, A.21 und A.23 (Abb. A.20 und A.22) anzeigen.

Wir fügen Abb. A.21 hinzu, um zu zeigen, wie die Tetragramme in der ursprünglichen Nachricht aussehen würden, wie sie vom deutschen Code-Schreiber geschrieben wurde. Wir stellen fest, dass die Methode der *Verschlüsselung* daraus immer noch unklar ist,

Abb. A.19 Achtzehn
Sequenzen lang

0	145	296	68	210	345	117	257	20	
nscg	enan	nte	nver	bren	nun	bser	las	sver	
166	317	87	222	376	143	298	67	205	
bren	nen	komm	adon	ich	imhi	nbl	ieka	ufku	
18	163	314	86	228	363	135	275	38	
ahru	ngni	eht	unbe	ding	ter	ford	erl	ichi	
184	335	105	240	16	161	316	85	223	
stun	dde	renk	enn	tnia	nahm	edu	rchu	nsur	
36	181	332	104	246	3	153	293	56	
onde	nzmi	tdi	eser	ges	andt	scha	ftp	unkt	
202	353	123	258	34	179	334	103	241	
absa	tza	usge	nom	menb	leib	env	orla	euf	
54	199	350	122	264	21	171	311	74	
ierz	ubez	ond	ersl	ist	enre	gist	era	bree	
220	371	141	276	52	197	352	121	259	
hnun	gen	quit	tun	genk	onto	bue	cher	und	
72	217	368	140	282	39	189	329	92	
bitt	esae	mtl	iche	mit	demk	rieg	ezu	samm	
238	11	159	294	70	215	370	139	277	
enh	aeng	ende	nko	mmav	oral	lem	also	die	
90	235	8	158	300	57	207	347	110	
tztu	ndf	uera	llez	uku	nftk	omma	auc	hnac	
256	29	177	312	88	233	10	157	295	
hfr	iede	nssc	hiu	ssko	mma	stre	ngst	ess	
108	253	26	176	318	75	225	365	128	
till	sch	weig	enzu	bew	ahre	nkla	mme	rzub	
274	47	195	330	106	251	28	175	313	
ezu	egli	chen	kem	main	ihr	enha	ende	nbe	
126	271	44	194	336	93	243	5	146	
romi	tti	eren	dode	rau	chnu	run	ange	nehm	
292	65	213	348	124	269	46	193	331	
sei	nkoe	nnte	kom	masc	for	tsor	gfae	lti	
144	289	62	212	354	111	261	23	164	
rera	ufb	ewnh	rung	gan	zbes	ond	erea	ufme	
310	83	231	366	142	287	64	211	349	
rks	amke	itzu	sch	enke	nis	tpun	ktab	sat	
162	307	80	230	372	129	279	41	182	
zbit	ted	iehi	nsic	htl	ichd	ers	chon	vcrh	
328	101	249	6	160	305	82	229	367	
and	enen	pap	iere	erfo	lgt	eaus	fueh	run	
180	325	98	248	12	147	297	59	200	
aufd	eng	ohei	mdi	erst	undd	end	iens	tder	
346	119	267	24	178	323	100	247	7	
ver	tret	eru	nser	esge	ner	alun	dad	mira	
198	343	116	266	30	165	315	77	218	
enft	ige	gezu	gna	hmen	hier	mit	alsp	qrbe	
364	137	285	42	196	341	118	265	25	
zei	chne	und	dess	euin	hal	tzus	amm	enmi	
216	361	134	284	48	183	333	95	236	
enoc	him	mern	ach	demb	esit	zes	oleh	erp	
4	155	303	60	214	359	136	283	43	
apie	rest	reb	ende	nfei	nde	fuer	uns	komp	
234	1	152	302	66	201	351	113	254	
ign	ochd	ieje	tzt	ncch	inbe	nut	zung	bef	
22	173	321	78	232	377	154	301	61	
indl	iohc	neh	iffr	ebue	che	rkom	mac	cdes	

Abb. A.20 Achtzehn
Sequenzen fortgesetzt

252	19	170	320	84	219	369	131	272	
lst	abes	klam	mer	aufu	eber	die	exis	ten	
	40	191	339	96	250	17	172	319	79
	zund	dasw	irk	ondi	eso	rver	tret	eri	stje
270	37	188	338	102	237	9	149	290	
sow	eite	rsow	iea	uchd	ieb	etre	ffen	deb	
	58	209	357	114	268	35	190	337	97
	rier	lich	eun	dtel	egr	aphi	acho	kor	resp
288	55	206	356	120	255	27	167	308	
und	gohe	imac	hri	ftst	ich	wcer	teru	nda	
	76	227	375	132	286	53	208	355	115
	nwei	sung	enk	omma	der	endu	rcha	uss	iche
306	73	224	374	138	273	45	185	326	
gdi	eses	auft	rag	essc	hri	ftli	chof	fen	
	94	245	15	150	304	71	226	373	133
	meld	enu	ndso	dann	auc	hden	vorl	ieg	ende
324	91	242	14	156	291	63	203	344	
gun	dres	tlo	sver	bren	nen	undd	ieas	che	
	112	263	33	168	322	89	244	13	151
	zers	chl	agen	punk	tab	satz	esg	ehoe	renh
342	109	260	32	174	309	81	221	362	
fin	dlic	hen	undi	nzuk	unf	tnoc	hans	ieg	
	130	281	51	186	340	107	262	31	169
	elan	gen	denp	apie	rek	omma	der	enau	fbew
360	127	278	50	192	327	99	239	2	
tdi	eser	bez	eieh	nung	sie	bitt	eim	geda	
	148	299	69	204	358	125	280	49	187
	eeht	nis	beha	lten	wol	lenp	unk	tseh	luss

da die Auswahl von Trigrammen mit einem nachfolgenden Leerzeichen im Vergleich zu Tetragrammen offensichtlich erst nach der Umordnung getroffen werden muss.

Zuerst zerlegen wir dies an Wortgrenzen, um Abb. A.24 zu erzeugen. Dann schauen wir uns das Deutsche genau an, um die Zeilen neu anzuordnen und die endgültige Nachricht von Abb. A.25 zu erzeugen. In dieser letzten Figur haben wir die Buchstaben, die im Original durcheinander waren, fett markiert. Die Verwirrungen stammen direkt von Yardley [1], und es ist nicht klar, ob sie in der ursprünglichen Nachricht waren oder ob diese Verwirrungen aus dem Druck des Buches stammen.

Schließlich bieten wir die Übersetzung von Yardley (S. 151–152) in Abb. A.26 an und als abschließenden Gedanken, für diejenigen, die Deutsch lesen können, bemerken wir, dass die ersten zehn Zeilen des Originaltextes den berühmten Kommentar von Mark Twain zu veranschaulichen scheinen:

Wann immer der literarische Deutsche in einen Satz eintaucht, ist das das Letzte, was Sie von ihm sehen werden, bis er auf der anderen Seite seines Atlantiks mit seinem Verb im Mund auftaucht.(*Ein Yankee aus Connecticut am Hofe König Arthurs*)

Abb. A.21 Achtzehn
Sequenzen, umgestellt

72	217	368	140	282	39	189	329	92	
bitt	esae	mtl	iche	mit	demk	rieg	ezu	samm	
238	11	159	294	70	215	370	139	277	
enh	aeng	ende	nko	mmav	oral	lem	also	die	
180	325	98	248	12	147	297	59	200	
aufd	eng	ohei	mdi	erst	undd	end	iens	tder	
346	119	267	24	178	323	100	247	7	
ver	tret	eru	nser	esge	ner	alun	dad	mira	
252	19	170	320	84	219	369	131	272	
lst	abes	klam	mer	aufu	eber	die	exis	ten	
40	191	339	96	250	17	172	319	79	
zund	dasw	irk	ondi	eso	rver	tret	eri	stje	
90	235	8	158	300	57	207	347	110	
tztu	ndf	uera	llez	uku	nftk	omma	auc	hnac	
256	29	177	312	88	233	10	157	295	
hfr	iede	nssc	hiu	ssko	mma	stre	ngst	ess	
108	253	26	176	318	75	225	365	128	
till	sch	weig	enzu	bew	ahre	nkla	mme	rzub	
274	47	195	330	106	251	28	175	313	
ezu	egli	chen	kem	main	ihr	enha	ende	nbe	
342	109	260	32	174	309	81	221	362	
fin	dlic	hen	undi	nzuk	unf	tnoc	hans	ieg	
130	281	51	186	340	107	262	31	169	
elan	gen	denp	apie	rek	omma	der	enau	fbew	
18	163	314	86	228	363	135	275	38	
ahru	ngni	eht	unbe	ding	ter	ford	erl	ichi	
184	335	105	240	16	161	316	85	223	
stun	dde	renk	enn	tnia	nahm	edu	rchu	nsur	
216	361	134	284	48	183	333	95	236	
enoc	him	mern	ach	demb	esit	zes	oleh	erp	
4	155	303	60	214	359	136	283	43	
apie	rest	reb	ende	nfei	nde	fuer	uns	komp	
126	271	44	194	336	93	243	5	146	
romi	tti	eren	dode	rau	chnu	run	ange	nehm	
292	65	213	348	124	269	46	193	331	
sei	nkoe	nnte	kom	masc	for	tsor	gfae	lti	
324	91	242	14	156	291	63	203	344	
gun	dres	tlo	sver	bren	nen	undd	ieas	che	
112	263	33	168	322	89	244	13	151	
zers	chl	agen	punk	tab	satz	esg	ehoe	renh	
54	199	350	122	264	21	171	311	74	
ierz	ubez	ond	ersl	ist	enre	gist	era	bree	
220	371	141	276	52	197	352	121	259	
hnun	gen	quit	tun	genk	onto	bue	cher	und	
270	37	188	338	102	237	9	149	290	
sow	eite	rsow	iea	uchd	ieb	etre	ffen	deb	
58	209	357	114	268	35	190	337	97	
rier	lich	eun	dtel	egr	aphi	acho	kor	resp	
36	181	332	104	246	3	153	293	56	
onde	nzmi	tdi	eser	ges	andt	scha	ftp	unkt	
202	353	123	258	34	179	334	103	241	
absa	tza	usge	nom	menb	leib	env	orla	euf	
234	1	152	302	66	201	351	113	254	
ign	ochd	ieje	tzt	ncch	inbe	nut	zung	bef	
22	173	321	78	232	377	154	301	61	
indl	iohc	neh	iffr	ebue	che	rkom	mac	cdes	

Abb. A.22 Achtzehn
Sequenzen umgestellt
(fortgesetzt)

288	55	206	356	120	255	27	167	308	
und	gohe	imac	hri	ftst	ich	wcer	teru	nda	
	76	227	375	132	286	53	208	355	115
	nwei	sung	enk	omma	der	endu	rcha	uss	iche
144	289	62	212	354	111	261	23	164	
rera	ufb	ewnh	rung	gan	zbes	ond	erea	ufme	
	310	83	231	366	142	287	64	211	349
	rks	amke	itzu	sch	enke	nis	tpun	ktab	sat
162	307	80	230	372	129	279	41	182	
zbit	ted	iehi	nsic	htl	ichd	ers	chon	vcrh	
	328	101	249	6	160	305	82	229	367
	and	enen	pap	iere	erfo	lgt	eaus	fueh	run
306	73	224	374	138	273	45	185	326	
gdi	eses	auft	rag	essc	hri	ftli	chof	fen	
	94	245	15	150	304	71	226	373	133
	meld	enu	ndso	dann	auc	hden	vorl	ieg	ende
0	145	296	68	210	345	117	257	20	
nscg	enan	nte	nver	bren	nun	bser	las	sver	
	166	317	87	222	376	143	298	67	205
	bren	nen	komm	adon	ich	imhi	nbl	ieka	ufku
198	343	116	266	30	165	315	77	218	
enft	ige	gezu	gna	hmen	hier	mit	alsp	qrbe	
	364	137	285	42	196	341	118	265	25
	zei	chne	und	dess	euin	hal	tzus	amm	enmi
360	127	278	50	192	327	99	239	2	
tdi	eser	bez	eieh	nung	sie	bitt	eim	geda	
	148	299	69	204	358	125	280	49	187
	eeht	nis	beha	lten	wol	lenp	unk	tseh	luss

bitt esae mtl iche mit demk rieg ezu samm enh aeng ende nko mmav oral lem also die
 aufd eng ohei mdi enst undd end iens tder ver tret eru nser esge ner alun dad mira
 lst abes klam mer aufo eber die exis ten zund dasw irk ondi eso rver tret eri stje
 tztu ndf uera llez uku nftk omma auc hnac hfr iede nssc hiu ssko mma stre ngst ess
 till sch weig enzu bew ahre nkla mme rzub ezu egli chen kem main ihr enha ende nbe
 fin dllic hen undi nzuk unf tnoc hans ieg elan gen denp apie rek omma der enau fbew
 ahru ngni eht unbe ding ter ford erl ichi stun dde renk enn tnia nahm edu rchu nsur
 enoc him mern ach demb esit zes oleh erp apie rest reb ende nfei nde fuer uns komp
 romi tti eren dode rau chnu run ange nehme sei nkoe nnte kom masc for tsor gfae lti
 gun dres tlo sver bren nen undd ieas che zers chl agen punk tab satz esg ehoe renh
 ierz ubez ond ersl ist enre gist era bree hnun gen quit tun genk onto bue cher und
 sow eite rsow iea uchd ieb etre ffen deb rier lich eun dtel egr aph i acho kor resp
 onde nzmi tdi eser ges andt scha ftp unkt absa tza usge nom menb leib env orla euf
 ign ochd ieje tzt nech inbe nut zung bef indl iohc neh iffir ebue che rkom mac cdes
 und gohe imac hri ftst ich weer teru nda nwei sung enk omma der endu rcha uss iche
 rera ufb ewnh rung gan zbes ond erea ufme rks amke itzu sch enke nis tpu n ktab sat
 zbit ted iehi nsic htl ichd ers chon vrh and enen pap iere erfo lgt eaus fueh run
 gdi eses auft rag essc hri ftli chof fen meld enu ndso dann auc hden vorl ieg ende
 nscg enan nte nver bren nun bser las sver bren nen komm adon ich imhi nbl ieka ufku
 enft ige gezu gna hmen hier mit als pqr bezeichne und desseu inhalt zusammen mi
 tdi eser bez eieh nung sie bitt eim geda eeht nis beha lten wol lenp unk tseh luss

Abb. A.23 Entschlüsselte Nachricht

bitte saemtliche mit dem kriege zusammen haengenden komma vorallem also die
 auf den goheimdienst und den dienst der vertreter unseres general und admira
 l stabes klammer auf ueber die existenz und das wirkondie sor vertreter istje
 tzt und fuer alle zukunft komma auch nach friedensschiuss komma strengstes s
 tillschweigen zu bewahren klammer zu bezueglichen kem main ihren haenden be
 findlichen und in zukunft noch ansiegelangenden papiere komma deren auf bew
 ahrung nicht unbedingterforderlichistundderen kenntnia nahme durch unsur
 e noch immer nach dem besitze s oleh er papieres trebenden feinde fuer uns komp
 romittieren dode rauch nur unangenehm sein koennte komma sc fort sorg faelti
 g und rest los verbrennen und die asche zerschlagen punkt absatz es gehoeren h
 ierzu bezonders listen register abrechnungen quittungen kontobuecher und
 soweiter sowie auch die betreffen debrierlich eund telegraphiacho korresp
 ondenz mit dieser gesandtschaft punkt absatz ausgenommen bleiben vorlaeuf
 ig noch die jetzt nech in benutzung befindl ioh cneh iffre buecher komma c cdes
 und goheim achriftstich weer ter und anweisungen komma deren durch aussiche
 r er auf bewnhrung ganz besondere aufmerksamkeit zu schenken ist punkt absat
 z bitte die hinsichtlich der schon vrhandenen papiere erfolgte aus fuehr un
 g dieses auftrages schriftlich offenmelden und so dann auch den vorliegende
 n scgenannten verbrennun bser lass verbrennen komma don ich im hinblik aufku
 enftige gezug nahmen hier mit als pqr bezeichne und desseu inhalt zusammen mi
 t dieser bezeichnung sie bitte im gedaehtnis behalten wollen punkt seh luss

Abb. A.24 Entschlüsselte Nachricht an Wortgrenzen unterbrochen

bitte saemtliche mit dem kriege zusammenhaengenden komma vorallem also die auf den geheimdienst und den dienst der vertreter unseres general und admiralstabes klammer auf ueber die existenz und das wirkon dieser vertreter ist jetzt und fuer alle zukunft komma auch nach friedensschluss komma strengstes stillschweigen zu bewahren klammer zu bezueglichen komma in ihren haenden befindlichen und in zukunft noch ansiegelangenden papiere komma deren aufbewahrung nicht unbedingt erforderlich ist und deren kenntnisnahme durch unsere noch immer nach dem besitze solcher papiere strebenden feinde fuer uns kompromittierende der auch nur unangenehm sein koennte komma so fort sorgfaeltig und restlos verbrennen und die asche zerschlagen punkt absatz es gehoeren hierzu besonders listen register abrechnungen quittungen kontobuecher und soweit sowie auch die betreffen debitorische und telegraphische korrespondenz mit dieser gesandtschaft punkt absatz ausgenommen bleiben vorlaeufig noch die jetzt noch in benutzung befindlichen chiffrebuecher komma codes und geheime schriftliche woerter und anweisungen komma deren durch aussichtser auf beanpruch ganz besondere aufmerksamkeit zu schenken ist punkt absatz bitte die hinsichtlich der schon verhandenen papiere erfolgte ausfuehrung dieses auftrages schriftlich offen melden und so dann auch den vorliegenden sogenannten verbrennungsbefehl verbrennen komma don ich im hinblick auf kuemftige gezuagnahmen hiermit als pqr bezeichne und dessen inhalt zusammen mit dieser bezeichnung sie bitte im gedaechnis behalten wollen punkt schluss

Abb. A.25 Endnachricht

Please carefully and immediately burn without remainder, and destroy the ashes of, all papers connected with the war, the preservation of which is not absolutely necessary, especially papers now in your hands or reaching you hereafter which have to do with the Secret Service and the service of the representatives of our General Staff and Admiralty Staff (strictest silence concerning the existence and activity of these representatives is to be observed now and for all future time, even after the conclusion of peace) which might be compromising or even unpleasant for us if they came to the knowledge of our enemies, who are still endeavoring to obtain possession of such papers.

Lists, registers, accounts, receipts, account books, etc., are especially included in these papers, as well as correspondence with this Embassy by letter and telegraph on the subjects mentioned.

Cipher books, codes and cipher keys and directions that are still in use are excepted for the present, and most particular attention must be paid to keeping them in absolute safety.

Please report in writing en claire the execution of this order so far as it relates to papers now on hand and then burn this so-called order for burning, which, for further reference, I herewith designate as PQR, and the contents of which together with this designation you will please retain in memory.

Abb. A.26 Die übersetzte Nachricht

AES Code

B.1 Einleitung

Dies ist eine Überarbeitung der Testvektoren und des Codes, die im Anhang von [3] als Anhänge B und C erscheinen.

Anhang B.2 von [3] hat eine Spur von Rijndael mit Klartextblöcken von 128 Bits und Schlüsselgröße 128. Wie dort angegeben (und hier leicht geändert), sind die Beschriftungen auf der Ausgabe

- r ist die Rundenzahl
- `input` ist die Eingabe für die Verschlüsselung
- `start` ist der Zustand (der 128-Bit-Block, der mit der Klartexteingabe beginnt und durch den Verschlüsselungsprozess verfolgt wird, um das Chifftrat zu ergeben) zu Beginn der Runde r
- `s_box` ist der Zustand nach der `s_box` Substitution
- `s_row` ist der Zustand nach der Shift-Row-Transformation
- `m_col` ist der Zustand nach der Mix-Column-Transformation
- `k_sch` ist der Schlüsselplanwert für Runde r
- `output` ist der Zustand nach der Verschlüsselung, das heißt, das Chifftrat.

B.2 Eine überarbeitete Anhang B.2

Dies ist im Wesentlichen das Gleiche wie Anhang B.2. Wir haben die Beschriftung leicht geändert, wir haben sowohl die Verschlüsselung als auch die Entschlüsselung nachverfolgt (anstatt nur die Verschlüsselung), und wir haben eine Übersetzung der Hex-Bytes in druckbare Zeichen aufgenommen. Das ENC Label steht für die Verschlüsselung des Klartextes, das DEC Label steht für die anschließende Entschlüsselung.

block length 128 key length 128

TEXT 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

KEY 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

ROUND 0 input 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

ROUND 0 k_sch 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

ROUND 1 start 19 3d e3 be a0 f4 e2 2b 9a c6 8d 2a e9 f8 48 08

ROUND 1 s_box d4 27 11 ae e0 bf 98 f1 b8 b4 5d e5 1e 41 52 30

ROUND 1 s_row d4 bf 5d 30 e0 b4 52 ae b8 41 11 f1 1e 27 98 e5

ROUND 1 m_col 04 66 81 e5 e0 cb 19 9a 48 f8 d3 7a 28 06 26 4c

ROUND 1 k_sch a0 fa fe 17 88 54 2c b1 23 a3 39 39 2a 6c 76 05

ROUND 2 start a4 9c 7f f2 68 9f 35 2b 6b 5b ea 43 02 6a 50 49

ROUND 2 s_box 49 de d2 89 45 db 96 f1 7f 39 87 1a 77 02 53 3b

ROUND 2 s_row 49 db 87 3b 45 39 53 89 7f 02 d2 f1 77 de 96 1a

ROUND 2 m_col 58 4d ca f1 1b 4b 5a ac db e7 ca a8 1b 6b b0 e5

ROUND 2 k_sch f2 c2 95 f2 7a 96 b9 43 59 35 80 7a 73 59 f6 7f

ROUND 3 start aa 8f 5f 03 61 dd e3 ef 82 d2 4a d2 68 32 46 9a

ROUND 3 s_box ac 73 cf 7b ef c1 11 df 13 b5 d6 b5 45 23 5a b8

ROUND 3 s_row ac c1 d6 b8 ef b5 5a 7b 13 23 cf df 45 73 11 b5

ROUND 3 m_col 75 ec 09 93 20 0b 63 33 53 c0 cf 7c bb 25 d0 dc

ROUND 3 k_sch 3d 80 47 7d 47 16 fe 3e 1e 23 7e 44 6d 7a 88 3b

ROUND 4 start 48 6c 4e ee 67 1d 9d 0d 4d e3 b1 38 d6 5f 58 e7

ROUND 4 s_box 52 50 2f 28 85 a4 5e d7 e3 11 c8 07 f6 cf 6a 94

ROUND 4 s_row 52 a4 c8 94 85 11 6a 28 e3 cf 2f d7 f6 50 5e 07

ROUND 4 m_col 0f d6 da a9 60 31 38 bf 6f c0 10 6b 5e b3 13 01

ROUND 4 k_sch ef 44 a5 41 a8 52 5b 7f b6 71 25 3b db 0b ad 00

ROUND 5 start e0 92 7f e8 c8 63 63 c0 d9 b1 35 50 85 b8 be 01

ROUND 5 s_box e1 4f d2 9b e8 fb fb ba 35 c8 96 53 97 6c ae 7c

ROUND 5 s_row e1 fb 96 7c e8 c8 ae 9b 35 6c d2 ba 97 4f fb 53

ROUND 5 m_col 25 d1 a9 ad bd 11 d1 68 b6 3a 33 8e 4c 4c c0 b0

ROUND 5 k_sch d4 d1 c6 f8 7c 83 9d 87 ca f2 b8 bc 11 f9 15 bc

ROUND 6 start f1 00 6f 55 c1 92 4c ef 7c c8 8b 32 5d b5 d5 0c

ROUND 6 s_box a1 63 a8 fc 78 4f 29 df 10 e8 3d 23 4c d5 03 fe

ROUND 6 s_row a1 4f 3d fe 78 e8 03 fc 10 d5 a8 df 4c 63 29 23

ROUND 6 m_col 4b 86 8d 6d 2c 4a 89 80 33 9d f4 e8 37 d2 18 d8

ROUND 6 k_sch 6d 88 a3 7a 11 0b 3e fd db f9 86 41 ca 00 93 fd

```

ROUND 7 start      26 0e 2e 17 3d 41 b7 7d e8 64 72 a9 fd d2 8b 25
ROUND 7 s_box      f7 ab 31 f0 27 83 a9 ff 9b 43 40 d3 54 b5 3d 3f
ROUND 7 s_row      f7 83 40 3f 27 43 3d f0 9b b5 31 ff 54 ab a9 d3
ROUND 7 m_col      14 15 b5 bf 46 16 15 ec 27 46 56 d7 34 2a d8 43
ROUND 7 k_sch      4e 54 f7 0e 5f 5f c9 f3 84 a6 4f b2 4e a6 dc 4f

ROUND 8 start      5a 41 42 b1 19 49 dc 1f a3 e0 19 65 7a 8c 04 0c
ROUND 8 s_box      be 83 2c c8 d4 3b 86 c0 0a e1 d4 4d da 64 f2 fe
ROUND 8 s_row      be 3b d4 fe d4 e1 f2 c8 0a 64 2c c0 da 83 86 4d
ROUND 8 m_col      00 51 2f d1 b1 c8 89 ff 54 76 6d cd fa 1b 99 ea
ROUND 8 k_sch      ea d2 73 21 b5 8d ba d2 31 2b f5 60 7f 8d 29 2f

ROUND 9 start      ea 83 5c f0 04 45 33 2d 65 5d 98 ad 85 96 b0 c5
ROUND 9 s_box      87 ec 4a 8c f2 6e c3 d8 4d 4c 46 95 97 90 e7 a6
ROUND 9 s_row      87 6e 46 a6 f2 4c e7 8c 4d 90 4a d8 97 ec c3 95
ROUND 9 m_col      47 37 94 ed 40 d4 e4 a5 a3 70 3a a6 4c 9f 42 bc
ROUND 9 k_sch      ac 77 66 f3 19 fa dc 21 28 d1 29 41 57 5c 00 6e

ROUND 10 start     eb 40 f2 1e 59 2e 38 84 8b a1 13 e7 1b c3 42 d2
ROUND 10 s_box     e9 09 89 72 cb 31 07 5f 3d 32 7d 94 af 2e 2c b5
ROUND 10 s_row     e9 31 7d b5 cb 32 2c 72 3d 2e 89 5f af 09 07 94
ROUND 10 k_sch     d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
ROUND 10 output    39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

ENC    39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

ROUND 10 output    39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32
ROUND 10 k_sch     d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
ROUND 10 s_row     e9 31 7d b5 cb 32 2c 72 3d 2e 89 5f af 09 07 94
ROUND 10 s_box     e9 09 89 72 cb 31 07 5f 3d 32 7d 94 af 2e 2c b5
ROUND 10 start     eb 40 f2 1e 59 2e 38 84 8b a1 13 e7 1b c3 42 d2

ROUND 9 k_sch      ac 77 66 f3 19 fa dc 21 28 d1 29 41 57 5c 00 6e
ROUND 9 m_col      47 37 94 ed 40 d4 e4 a5 a3 70 3a a6 4c 9f 42 bc
ROUND 9 s_row      87 6e 46 a6 f2 4c e7 8c 4d 90 4a d8 97 ec c3 95
ROUND 9 s_box      87 ec 4a 8c f2 6e c3 d8 4d 4c 46 95 97 90 e7 a6
ROUND 9 start      ea 83 5c f0 04 45 33 2d 65 5d 98 ad 85 96 b0 c5

ROUND 8 k_sch      ea d2 73 21 b5 8d ba d2 31 2b f5 60 7f 8d 29 2f
ROUND 8 m_col      00 51 2f d1 b1 c8 89 ff 54 76 6d cd fa 1b 99 ea
ROUND 8 s_row      be 3b d4 fe d4 e1 f2 c8 0a 64 2c c0 da 83 86 4d
ROUND 8 s_box      be 83 2c c8 d4 3b 86 c0 0a e1 d4 4d da 64 f2 fe
ROUND 8 start      5a 41 42 b1 19 49 dc 1f a3 e0 19 65 7a 8c 04 0c

```

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

```
DEC  32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
```

B.3 Eine überarbeitete Anhang B.3

Dies ist im Wesentlichen das Gleiche wie Anhang [B.3](#).

Wir haben die Beschriftung leicht geändert, wir haben sowohl die Verschlüsselung als auch die Entschlüsselung nachverfolgt (anstatt nur die Verschlüsselung), und wir haben eine Übersetzung der Hex-Bytes in druckbare Zeichen aufgenommen. Das ENC Etikett ist für die Verschlüsselung des Klartextes, ENC Etikett ist für die anschließende wiederholte Verschlüsselung.

block length 128 key length 128

TEXT 00000000000000000000000000000000
KEY 00000000000000000000000000000000
ENC 66e94bd4ef8a2c3b884cfa59ca342b2e
ENC f795bd4a52e29ed713d313fa20e98dbc

block length 160 key length 128

TEXT 00000000000000000000000000000000
KEY 00000000000000000000000000000000
ENC 9e38b8eb1d2025a1665ad4b1f5438bb5cae1ac3f
ENC 939c167e7f916d45670ee21bfc939e1055054a96

block length 192 key length 128

TEXT 00000000000000000000000000000000
KEY 00000000000000000000000000000000
ENC a92732eb488d8bb98ecd8d95dc9c02e052f250ad369b3849
ENC 106f34179c3982ddc6750aa01936b7a180e6b0b9d8d690ec

block length 224 key length 128

TEXT 00000000000000000000000000000000
KEY 00000000000000000000000000000000
ENC 0623522d88f7b9c63437537157f625dd5697ab628a3b9be2549895c8
ENC 93f93cbdabe23415620e6990b0443d621f6afbd6edefd6990a1965a8

block length 256 key length 128

TEXT 00000000000000000000000000000000
KEY 00000000000000000000000000000000
ENC a693b288df7dae5b1757640276439230db77c4cd7a871e24d6162e54af434891
ENC 5f05857c80b68ea42ccbc759d42c28d5cd490f1d180c7a9397ee585bea770391

block length 128 key length 160

TEXT 00000000000000000000000000000000
KEY 00000000000000000000000000000000
ENC 94b434f8f57b9780f0eff1a9ec4c112c
ENC 35a00ec955df43417ceac2ab2b3f3e76

block length 160 key length 160

TEXT 00000000000000000000000000000000
KEY 00000000000000000000000000000000
ENC 33b12ab81db7972e8fdc529dda46fcb529b31826
ENC 97f03eb018c0bb9195bf37c6a0aece8e4cb8de5f

block length 192 key length 160

```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC  528e2fff6005427b67bb1ed31ecc09a69ef41531df5ba5b2
ENC  71c7687a4c93ebc35601e3662256e10115beed56a410d7ac
```

```
block length 224 key length 160
```

```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY 0000000000000000000000000000000000000000000000000000000000000000
ENC 58a0c53f3822a32464704d409c2fd0521f3a93e1f6fcfd4c87f1c551
ENC d8e93ef2eb49857049d6f6e0f40b67516d2696f94013c065283f7f01
```

```
block length 256 key length 160
```

[illegible]

```
block length 128 key length 192
```

```
TEXT 00000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC  aae06992acbf52a3e8f4a96ec9300bd7
ENC  52f674b7b9030fdab13d18dc214eb331
```

```
block length 160 key length 192
```

```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC  33060f9d4705ddd2c7675f0099140e5a98729257
ENC  012cab64982156a5710e790f85ec442ce13c520f
```

```
block length 192 key length 192
```

```
TEXT 00000000000000000000000000000000000000000000000000000000000000000000
KEY  00000000000000000000000000000000000000000000000000000000000000000000
ENC  c6348be20007bac4a8bd62890c8147a2432e760e9a9f9ab8
ENC  eb9def13c253f81c1fc2829426ed166a65a105c6a04ca33d
```

```
block length 224 key length 192
```

```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC  3856b1b7bea77c4611e3397066828aadda04706a2c8009df40a811fe
ENC  160ad76a97ae2c1e05942fde3da2962684a92ccc74b8dc23bde4f469
```

```
block length 256 key length 192
```

```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC f927363ef5b3b4984a9eb9109844152ec167f08102644e3f9028070433df9f2a
```

ENC 4e03389c68b2e3f623ad8f7f6bfc88613b86f334f4148029ae25f50db144b80c

block length 128 key length 224

TEXT 00000000000000000000000000000000

KEY 00

ENC 73f8dff62a36f3ebf31d6f73a56ff279

ENC 3a72f21e10b6473ea9ff14a232e675b4

block length 160 key length 224

TEXT 000

KEY 00

ENC e9f5ea0fa39bb6ad7339f28e58e2e7535f261827

ENC 06ef9bc82905306d45810e12d0807796a3d338f9

block length 192 key length 224

TEXT 000

KEY 00

ENC ecbe9942cd6703e16d358a829d542456d71bd3408eb23c56

ENC fd10458ed034368a34047905165b78a6f0591ffeebf47cc7

block length 224 key length 224

TEXT 000

KEY 00

ENC fe1cf0c8ddad24e3d751933100e8e89b61cd5d31c96abff7209c495c

ENC 515d8e2f2b9c5708f112c6de31caca47afb86838b716975a24a09cd4

block length 256 key length 224

TEXT 000

KEY 00

ENC bc18bf6d369c955bbb271cbcd66c368356dba5b33c0005550d2320b1c617e21

ENC 60aba1d2be45d8abfdcf97bcb39f6c17df29985cf321bab75e26a26100ac00af

block length 128 key length 256

TEXT 000

KEY 00

ENC dc95c078a2408989ad48a21492842087

ENC 08c374848c228233c2b34f332bd2e9d3

block length 160 key length 256

TEXT 000

KEY 00

ENC 30991844f72973b3b2161f11e7f8d9863c5118

ENC eef8b7cc9dbe0f03alfe9d82e9a759fd281c67e0

block length 192 key length 256


```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC  17004e806faef168fc9cd56f98f070982075c70c8132b945
ENC  bed33b0af364dbf15f9c2f3fb24fbdf1d36129c586eea6b7
```

block length 224 key length 256

```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC  9bf26fad5680d56b572067ec2fe162f449404c86303f8be38fab6e02
ENC  658f144a34af44aae66cfdab955c483dfbcb4ee9a19a6701f158a66
```

block length 256 key length 256

```
TEXT 0000000000000000000000000000000000000000000000000000000000000000
KEY  0000000000000000000000000000000000000000000000000000000000000000
ENC  c6227e7740b7e53b5cb77865278eab0726f62366d9aabad908936123a1fc8af3
ENC  9843e807319c32ad1ea3935ef56a2ba96e4bf19c30e47d88a2b97cbbf2e159e7
```

B.4 Eine überarbeitete Anhang C

Dies ist im Wesentlichen das Gleiche wie Anhang C, aber wir haben die Formatierung des Codes leicht überarbeitet. Die größte Änderung besteht darin, dass die Decrypt Funktion in Anhang C falsch ist, da sie die Reihenfolge der Aufrufe der Verschlüsselungsschritte beibehält, anstatt die Reihenfolge umzukehren. Zusätzlich zur Korrektur dieses Fehlers haben wir Aufrufe zu Funktionen hinzugefügt, die die Rückverfolgungsinformationen von Anhang [B.2](#) ausgeben.

B.4.1 AES Funktionen

Die in AES verwendeten Funktionen werden hier angezeigt.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

typedef unsigned char word8;
typedef unsigned int word32;

#define MAXBC 8
#define MAXKC 8
#define MAXROUNDS 14

bool testd2, testd3, testttext;
int BC, KC, ROUNDS;

word8 Logtable[256] = {
    0,  0,  25,  1,  50,  2,  26, 198,  75, 199,  27, 104,  51, 238, 223,  3,
  100,  4, 224, 14,  52, 141, 129, 239,  76, 113,  8, 200, 248, 105,  28, 193,
  125, 194,  29, 181, 249, 185,  39, 106,  77, 228, 166, 114, 154, 201,  9, 120,
  101,  47, 138,  5,  33,  15, 225,  36,  18, 240, 130,  69,  53, 147, 218, 142,
  150, 143, 219, 189,  54, 208, 206, 148,  19,  92, 210, 241,  64,  70, 131,  56,
  102, 221, 253,  48, 191,  6, 139,  98, 179,  37, 226, 152,  34, 136, 145,  16,
  126, 110,  72, 195, 163, 182,  30,  66,  58, 107,  40,  84, 250, 133,  61, 186,
  43, 121,  10,  21, 155, 159,  94, 202,  78, 212, 172, 229, 243, 115, 167,  87,
  175,  88, 168,  80, 244, 234, 214, 116,  79, 174, 233, 213, 231, 230, 173, 232,
  44, 215, 117, 122, 235,  22,  11, 245,  89, 203,  95, 176, 156, 169,  81, 160,
  127,  12, 246, 111,  23, 196,  73, 236, 216,  67,  31,  45, 164, 118, 123, 183,
  204, 187,  62,  90, 251,  96, 177, 134,  59,  82, 161, 108, 170,  85,  41, 157,
  151, 178, 135, 144,  97, 190, 220, 252, 188, 149, 207, 205,  55,  63,  91, 209,
  83,  57, 132,  60,  65, 162, 109,  71,  20,  42, 158,  93,  86, 242, 211, 171,
  68,  17, 146, 217,  35,  32,  46, 137, 180, 124, 184,  38, 119, 153, 227, 165,
  103,  74, 237, 222, 197,  49, 254,  24,  13,  99, 140, 128, 192, 247, 112,  7,
};

word8 Alogtable[256] = {
  1,  3,  5, 15, 17,  51,  85, 255,  26,  46, 114, 150, 161, 248, 19,  53,
  95, 225,  56,  72, 216, 115, 149, 164, 247,  2,  6,  10,  30,  34, 102, 170,
  229,  52,  92, 228,  55,  89, 235,  38, 106, 190, 217, 112, 144, 171, 230,  49,
  83, 245,  4,  12,  20,  60,  68, 204,  79, 209, 104, 184, 211, 110, 178, 205,
  76, 212, 103, 169, 224,  59,  77, 215,  98, 166, 241,  8,  24,  40, 120, 136,
  131, 158, 185, 208, 107, 189, 220, 127, 129, 152, 179, 206,  73, 219, 118, 154,
  181, 196,  87, 249,  16,  48,  80, 240,  11,  29,  39, 105, 187, 214,  97, 163,
  254,  25,  43, 125, 135, 146, 173, 236,  47, 113, 147, 174, 233,  32,  96, 160,

```

```

251, 22, 58, 78, 210, 109, 183, 194, 93, 231, 50, 86, 250, 21, 63, 65,
195, 94, 226, 61, 71, 201, 64, 192, 91, 237, 44, 116, 156, 191, 218, 117,
159, 186, 213, 100, 172, 239, 42, 126, 130, 157, 188, 223, 122, 142, 137, 128,
155, 182, 193, 88, 232, 35, 101, 175, 234, 37, 111, 177, 200, 67, 197, 84,
252, 31, 33, 99, 165, 244, 7, 9, 27, 45, 119, 153, 176, 203, 70, 202,
69, 207, 74, 222, 121, 139, 134, 145, 168, 227, 62, 66, 198, 81, 243, 14,
18, 54, 90, 238, 41, 123, 141, 140, 143, 138, 133, 148, 167, 242, 13, 23,
57, 75, 221, 124, 132, 151, 162, 253, 28, 36, 108, 180, 199, 82, 246, 1,
};

```

```

word8 S[256] = {
99, 124, 119, 123, 242, 107, 111, 197, 48, 1, 103, 43, 254, 215, 171, 118,
202, 130, 201, 125, 250, 89, 71, 240, 173, 212, 162, 175, 156, 164, 114, 192,
183, 253, 147, 38, 54, 63, 247, 204, 52, 165, 229, 241, 113, 216, 49, 21,
4, 199, 35, 195, 24, 150, 5, 154, 7, 18, 128, 226, 235, 39, 178, 117,
9, 131, 44, 26, 27, 110, 90, 160, 82, 59, 214, 179, 41, 227, 47, 132,
83, 209, 0, 237, 32, 252, 177, 91, 106, 203, 190, 57, 74, 76, 88, 207,
208, 239, 170, 251, 67, 77, 51, 133, 69, 249, 2, 127, 80, 60, 159, 168,
81, 163, 64, 143, 146, 157, 56, 245, 188, 182, 218, 33, 16, 255, 243, 210,
205, 12, 19, 236, 95, 151, 68, 23, 196, 167, 126, 61, 100, 93, 25, 115,
96, 129, 79, 220, 34, 42, 144, 136, 70, 238, 184, 20, 222, 94, 11, 219,
224, 50, 58, 10, 73, 6, 36, 92, 194, 211, 172, 98, 145, 149, 228, 121,
231, 200, 55, 109, 141, 213, 78, 169, 108, 86, 244, 234, 101, 122, 174, 8,
186, 120, 37, 46, 28, 166, 180, 198, 232, 221, 116, 31, 75, 189, 139, 138,
112, 62, 181, 102, 72, 3, 246, 14, 97, 53, 87, 185, 134, 193, 29, 158,
225, 248, 152, 17, 105, 217, 142, 148, 155, 30, 135, 233, 206, 85, 40, 223,
140, 161, 137, 13, 191, 230, 66, 104, 65, 153, 45, 15, 176, 84, 187, 22,
};

```

```

word8 Si[256] = {
82, 9, 106, 213, 48, 54, 165, 56, 191, 64, 163, 158, 129, 243, 215, 251,
124, 227, 57, 130, 155, 47, 255, 135, 52, 142, 67, 68, 196, 222, 233, 203,
84, 123, 148, 50, 166, 194, 35, 61, 238, 76, 149, 11, 66, 250, 195, 78,
8, 46, 161, 102, 40, 217, 36, 178, 118, 91, 162, 73, 109, 139, 209, 37,
114, 248, 246, 100, 134, 104, 152, 22, 212, 164, 92, 204, 93, 101, 182, 146,
108, 112, 72, 80, 253, 237, 185, 218, 94, 21, 70, 87, 167, 141, 157, 132,
144, 216, 171, 0, 140, 188, 211, 10, 247, 228, 88, 5, 184, 179, 69, 6,
208, 44, 30, 143, 202, 63, 15, 2, 193, 175, 189, 3, 1, 19, 138, 107,
58, 145, 17, 65, 79, 103, 220, 234, 151, 242, 207, 206, 240, 180, 230, 115,
150, 172, 116, 34, 231, 173, 53, 133, 226, 249, 55, 232, 28, 117, 223, 110,
71, 241, 26, 113, 29, 41, 197, 137, 111, 183, 98, 14, 170, 24, 190, 27,
252, 86, 62, 75, 198, 210, 121, 32, 154, 219, 192, 254, 120, 205, 90, 244,
31, 221, 168, 51, 136, 7, 199, 49, 177, 18, 16, 89, 39, 128, 236, 95,
96, 81, 127, 169, 25, 181, 74, 13, 45, 229, 122, 159, 147, 201, 156, 239,
160, 224, 59, 77, 174, 42, 245, 176, 200, 235, 187, 60, 131, 83, 153, 97,

```

```

    23, 43, 4, 126, 186, 119, 214, 38, 225, 105, 20, 99, 85, 33, 12, 125,
};

word32 RC[30] = {0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
                 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
                 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
                 0xfa, 0xef, 0xc5};

static word8 shifts[5][4] = {{0, 1, 2, 3},
                              {0, 1, 2, 3},
                              {0, 1, 2, 3},
                              {0, 1, 2, 4},
                              {0, 1, 3, 4}};

static int numrounds[5][5] = {{10, 11, 12, 13, 14},
                              {11, 11, 12, 13, 14},
                              {12, 12, 12, 13, 14},
                              {13, 13, 13, 13, 14},
                              {14, 14, 14, 14, 14}};

/*****
 * Multiply two elements of GF(256)
 * Required for MixColumns and InvMixColumns
 **/
word8 mul(word8 a, word8 b) {
    if (a && b) return Alogtable[(Logtable[a] + Logtable[b])%255];
    else return 0;
}

/*****
 * XOR corresponding text input and round key input bytes
 **/
void AddRoundKey(word8 a[4][MAXBC], word8 rk[4][MAXBC]) {
    int i, j;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] ^= rk[i][j];
        }
    }
}

/*****
 * Replace every byte of the input by the byte at that place
 * in the non-linear S-box
 **/

```

```

void SubBytes(word8 a[4][MAXBC], word8 box[255]) {
    int i, j;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] = box[a[i][j]];
        }
    }
}

/*****
* Row 0 remains unchanged.
* The other three rows are shifted a variable amount.
**/
void ShiftRows(word8 a[4][MAXBC], word8 d) {
    word8 tmp[MAXBC];
    int i, j;

    if (d == 0) {
        for (i = 1; i < 4; i++) {
            for (j = 0; j < BC; j++) {
                tmp[j] = a[i][(j + shifts[BC-4][i]) % BC];
            }
            for (j = 0; j < BC; j++) {
                a[i][j] = tmp[j];
            }
        }
    }
    else {
        for (i = 1; i < 4; i++) {
            for (j = 0; j < BC; j++) {
                tmp[j] = a[i][(BC + j - shifts[BC-4][i]) % BC];
            }
            for (j = 0; j < BC; j++) {
                a[i][j] = tmp[j];
            }
        }
    }
}

/*****
* Mix the four bytes of every column in a linear way.
**/
void MixColumns(word8 a[4][MAXBC]) {
    word8 b[4][MAXBC];
    int i, j;

```

```

    for (j = 0; j < BC; j++) {
        for (i = 0; i < 4; i++) {
            b[i][j] = mul(2, a[i][j])
                ^ mul(3, a[(i+1)%4][j])
                ^ a[(i+2)%4][j]
                ^ a[(i+3)%4][j];
        }
    }
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] = b[i][j];
        }
    }
}

/*****
* Mix the four bytes of every column in a linear way.
* This is the opposite operation of MixColumns.
**/
void InvMixColumns(word8 a[4][MAXBC]) {
    word8 b[4][MAXBC];
    int i, j;

    for (j = 0; j < BC; j++) {
        for (i = 0; i < 4; i++) {
            b[i][j] = mul(0xe, a[i][j])
                ^ mul(0xb, a[(i+1)%4][j])
                ^ mul(0xd, a[(i+2)%4][j])
                ^ mul(0x9, a[(i+3)%4][j]);
        }
    }
    for (i = 0; i < 4; i++) {
        for (j = 0; j < BC; j++) {
            a[i][j] = b[i][j];
        }
    }
}

/*****
*
**/
int KeyExpansion(word8 k[4][MAXKC], word8 W[MAXROUNDS+1][4][MAXBC]) {
    // Calculate the required round keys.
    int i, j, t, RCpointer = 1;

```

```

word8 tk[4][MAXKC];

for (j = 0; j < KC; j++) {
    for (i = 0; i < 4; i++) {
        tk[i][j] = k[i][j];
    }
}

t = 0;
// copy values into round key array
for (j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++) {
    for (i = 0; i < 4; i++) {
        W[t / BC][i][t % BC] = tk[i][j];
    }
}

while (t < (ROUNDS+1)*BC) {
    // while not enough round key material calculated, calc new values
    for (i = 0; i < 4; i++) {
        tk[i][0] ^= S[tk[(i+1)%4][KC-1]];
    }
    tk[0][0] ^= RC[RCpointer++];

    if (KC <= 6) {
        for (j = 1; j < KC; j++) {
            for (i = 0; i < 4; i++) {
                tk[i][j] ^= tk[i][j-1];
            }
        }
    } // if (KC <= 6)
    else {
        for (j = 1; j < 4; j++) {
            for (i = 0; i < 4; i++) {
                tk[i][j] ^= tk[i][j-1];
            }
        }
        for (i = 0; i < 4; i++) {
            tk[i][4] ^= S[tk[i][3]];
        }
        for (j = 5; j < KC; j++) {
            for (i = 0; i < 4; i++) {
                tk[i][j] ^= tk[i][j-1];
            }
        }
    } // else
}

```

```

    // copy values into round key array
    for (j = 0; (j < KC) && (t < (ROUNDS+1)*BC); j++, t++) {
        for (i = 0; i < 4; i++) {
            W[t / BC][i][t % BC] = tk[i][j];
        }
    }
} // while (t < (ROUNDS+1)*BC) {

return 0;
} // int KeyExpansion(word8 k[][[]], word8 W[][[]]) {

/*****
* Encryption of one block.
**/
int Encrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]) {
    int r;

    dumpvaluesBC(0, "input ", a, 4, BC, testd2);

    // Begin with a key addition.
    AddRoundKey(a, rk[0]);

    dumpvaluesKC(0, "k_sch ", rk[0], 4, KC, testd2);
    if (testd2) printf("\n");
    dumpvaluesBC(1, "start ", a, 4, BC, testd2);

    // ROUNDS-1 ordinary rounds.
    for (r = 1; r < ROUNDS; r++) {
        SubBytes(a, S);
        dumpvaluesBC(r, "s_box ", a, 4, BC, testd2);

        ShiftRows(a, 0);
        dumpvaluesBC(r, "s_row ", a, 4, BC, testd2);

        MixColumns(a);
        dumpvaluesBC(r, "m_col ", a, 4, BC, testd2);

        AddRoundKey(a, rk[r]);
        dumpvaluesKC(r, "k_sch ", rk[r], 4, KC, testd2);
        if (testd2) printf("\n");
        dumpvaluesBC(r+1, "start ", a, 4, BC, testd2);
    }

    // Last round is special: there is no MixColumns.

```



```

    SubBytes(a, S);
    dumpvaluesBC(r, "s_box ", a, 4, BC, testd2);

    ShiftRows(a, 0);
    dumpvaluesBC(r, "s_row ", a, 4, BC, testd2);
    dumpvaluesKC(r, "k_sch ", rk[r], 4, KC, testd2);

    AddRoundKey(a, rk[ROUNDS]);
    dumpvaluesBC(r, "output", a, 4, BC, testd2);

    return 0;
}

/*****
* To decrypt:
*   Apply the inverse operations of the encrypt routine,
*   in opposite order.
*
*   AddRoundKey is equal to its inverse.
*   The inverse of SubBytes with table S is
*       SubBytes with the inverse table Si.
*   The inverse of Shiftrows is Shiftrows over
*       a suitable distance.
**/
int Decrypt(word8 a[4][MAXBC], word8 rk[MAXROUNDS+1][4][MAXBC]) {
    int r;

    // First the special round:
    //   without InvMixColumns
    //   with extra AddRoundKey

    dumpvaluesBC(ROUNDS, "output", a, 4, BC, testd2);
    AddRoundKey(a, rk[ROUNDS]);

    dumpvaluesKC(ROUNDS, "k_sch ", rk[ROUNDS], 4, KC, testd2);
    dumpvaluesBC(ROUNDS, "s_row ", a, 4, BC, testd2);
    // This was the original order of the functions.
    //   SubBytes(a, Si);
    //   dumpvaluesBC(ROUNDS, "s_box ", a, 4, 4, testd2);
    //   ShiftRows(a, 1);

    // This is the revised order of the functions.
    // This order works and the original one does not.
    ShiftRows(a, 1);

```

```

    dumpvaluesBC(ROUNDS, "s_box ", a, 4, BC, testd2);
    SubBytes(a, Si);

    // ROUNDS-1 ordinary rounds.
    for (r = ROUNDS-1; r > 0; r--) {
        dumpvaluesBC(r+1, "start ", a, 4, BC, testd2);
        if (testd2) printf("\n");
        AddRoundKey(a, rk[r]);

        dumpvaluesKC(r, "k_sch ", rk[r], 4, KC, testd2);
        dumpvaluesBC(r, "m_col ", a, 4, BC, testd2);
        InvMixColumns(a);

        dumpvaluesBC(r, "s_row ", a, 4, BC, testd2);
        // This was the original order of the functions.
        // SubBytes(a, Si);
        // ShiftRows(a, 1);

        // This is the revised order of the functions.
        // This order works and the original one does not.
        ShiftRows(a, 1);
        dumpvaluesBC(r, "s_box ", a, 4, BC, testd2);

        SubBytes(a, Si);
    }

    dumpvaluesBC(r+1, "start ", a, 4, BC, testd2);
    if (testd2) printf("\n");
    dumpvaluesKC(0, "k_sch ", rk[0], 4, KC, testd2);

    AddRoundKey(a, rk[0]);

    dumpvaluesBC(0, "input ", a, 4, BC, testd2);

    return 0;
}

```

B.4.2 AES Hauptprogramm

Das Hauptprogramm, das für die Daten in Anhang [B.2](#) und [B.3](#) verwendet wird, wird hier angezeigt.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include "aesutils.c"
#include "aesfunctions.c"

/*****
 *
 **/
int main() {
    word8 a[4][MAXBC], rk[MAXROUNDS+1][4][MAXBC], sk[4][MAXKC];

    char* key;
    char* text;
    int bcupper = 4; // default value
    int kcupper = 4; // default value

    testd2 = false;
    testd3 = false;
    testtext = false;

#ifdef TESTD2
    testd2 = true;
#elif TESTD3
    testd3 = true;
#elif TESTTEXT
    testtext = true;
#else
#error "Must define TESTD2 or TESTD3 or TESTTEXT"
#endif

    text = readstuff("xtext.txt");
    key = readstuff("xkey.txt");

    if (testd3) {
        bcupper = 8;
        kcupper = 8;
    }
    else if (testd2) {
        bcupper = 4;
        kcupper = 4;
    }
```

```

    }
    else if (testtext) {
        bcupper = 4;
        kcupper = 8;
    }

    for (KC = 4; KC <= kcupper; KC++) {
        for (BC = 4; BC <= bcupper; BC++) {
            ROUNDS = numrounds[KC-4][BC-4];

            if (testd3) {
                filltextallzeros(a);
                fillkeyallzeros(sk);
            }
            else if (testd2) {
                filltextd2(a);
                fillkeyd2(sk);
            }
            else if (testtext) {
                filltexttest(a);
                fillkeytest(sk);
            }
            else {
                printf("ERROR testd2 testd3\n");
                exit(0);
            }

            KeyExpansion(sk, rk);

#ifdef KEYSCHED
            // Print key schedule.
            printf("KEY SCHEDULE\n");
            if ((KC == 4) && (BC == 4)) {
                for (int r = 0; r < ROUNDS+1; r++) {
                    printf("%2d", r);
                    for (j = 0; j < 4; j++) {
                        for (i = 0; i < 4; i++) {
                            printf(" %02X", rk[r][i][j]);
                        }
                    }
                    printf("\n");
                }
            }
            printf("\n");
        }
    }
#endif

```

```

printf("block length %d  key length %d\n", 32*BC, 32*KC);
dump2dcolsBC("TEXT", a, 4, BC);
dump2dcolsBCchar("TEXT", a, 4, BC);
printf("\n");

dump2dcolsKC("KEY ", sk, 4, KC);
dump2dcolsKCchar("KEY ", sk, 4, KC);
printf("\n");

Encrypt(a, rk);
printf("\n");
dump2dcolsBC("ENC ", a, 4, BC);
dump2dcolsBCchar("CHAR", a, 4, BC);
printf("\n");

if (testd2 || testtext) {
    Decrypt(a, rk);
    printf("\n");
    dump2dcolsBC("DEC ", a, 4, BC);
    dump2dcolsBCchar("CHAR", a, 4, BC);
    printf("\n");
} // if (testd2) {
else if (testd3) {
    Encrypt(a, rk);
    dump2dcolsBC("DEC ", a, 4, BC);
    dump2dcolsBCchar("CHAR", a, 4, BC);
    printf("\n");
} // else if (testd3) {
} // for (BC = 4; BC <= 8; BC++) {
} // for (KC = 4; KC <= 8; KC++) {
}

```

B.4.3 AES Eingabe/Ausgabe Hilfsprogramme

Wir zeigen hier einige lokal erstellte Hilfsfunktionen. Ja, das sind Hacks. Wir entschuldigen uns nicht wirklich, obwohl wir es vielleicht sollten.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef unsigned char word8;
typedef unsigned int word32;

#define MAXBC 8
#define MAXKC 8
#define MAXROUNDS 14

int BC, KC, ROUNDS;

/*****
*
**/
void dump2drowsBC(char* label, word8 thevalues[4][MAXBC],
                  int limitrow, int limitcol) {
    int i, j;
    printf("%s by rows\n", label);
    for (i = 0; i < limitrow; i++) {
        printf("%2d", i);
        for (j = 0; j < limitcol; j++) {
            printf(" %02x", thevalues[i][j]);
        }
        printf("\n");
    }
}

/*****
*
**/
void dump2drowsKC(char* label, word8 thevalues[4][MAXKC],
                  int limitrow, int limitcol) {
    int i, j;
    printf("%s by rows\n", label);
    for (i = 0; i < limitrow; i++) {
        printf("%2d", i);
        for (j = 0; j < limitcol; j++) {
            printf(" %02x", thevalues[i][j]);
        }
        printf("\n");
    }
}

```

```

}

/*****
*
**/
void dump2dcolsBC(char* label, word8 thevalues[4][MAXBC], int limitrow,
                  int limitcol) {
    int i, j;

    // print as one row
    printf("%s by cols ", label);
    for (j = 0; j < limitcol; j++) {
        for (i = 0; i < limitrow; i++) {
            printf(" %02x", thevalues[i][j]);
        }
    }
    printf("\n");
}

/*****
*
**/
void dump2dcolsBCchar(char* label, word8 thevalues[4][MAXBC], int limitrow,
                      int limitcol) {
    int i, j;

    // print as one row
    printf("%s by cols ", label);
    for (j = 0; j < limitcol; j++) {
        for (i = 0; i < limitrow; i++) {
            if ((thevalues[i][j] >= 0x21) && (thevalues[i][j] <= 0x7d)) {
                printf(" %2c", thevalues[i][j]);
            }
            else {
                printf("  ~");
            }
        }
    }
    printf("\n");
}

/*****
*
**/
void dump2dcolsKC(char* label, word8 thevalues[4][MAXKC], int limitrow,

```

```

        int limitcol) {

    int i, j;

    // print as one row
    printf("%s by cols ", label);
    for (j = 0; j < limitcol; j++) {
        for (i = 0; i < limitrow; i++) {
            printf(" %02x", thevalues[i][j]);
        }
    }
    printf("\n");
}

/*****
*
**/
void dump2dcolsKCchar(char* label, word8 thevalues[4][MAXKC], int limitrow,
                    int limitcol) {

    int i, j;

    // print as one row
    printf("%s by cols ", label);
    for (j = 0; j < limitcol; j++) {
        for (i = 0; i < limitrow; i++) {
            if ((thevalues[i][j] >= 0x21) && (thevalues[i][j] <= 0x7d)) {
                printf(" %2c", thevalues[i][j]);
            }
            else {
                printf("  ~");
            }
        }
    }
    printf("\n");
}

/*****
*
**/
void dump3d(word8 thevalues[MAXROUNDS+1][4][MAXBC],
            int limitx, int limity, int limitz) {

    int i, j, k;
    for (i = 0; i < limitx; i++) {
        for (j = 0; j < limity; j++) {
            printf("%2d %2d", i, j);
            for (k = 0; k < limitz; k++) {

```



```

        printf(" %02x", thevalues[i][j][k]);
    }
    printf("\n");
}
printf("\n");
}
}

/*****
*
**/
void dump3dcols(word8 thevalues[MAXROUNDS+1][4][MAXBC],
               int limiti, int limitj, int limitk) {
    int i, j, k;
    for (i = 0; i < limiti; i++) {
        printf("%2d ", i);
        for (k = 0; k < limitk; k++) {
            for (j = 0; j < limitj; j++) {
                printf(" %02x", thevalues[i][j][k]);
            }
        }
        printf("\n");
    }
}

/*****
*
**/
void dumpvaluesBC(int round, char* label, word8 thevalues[4][MAXBC],
                 int limitrow, int limitcol, bool printflag) {
    if (printflag) {
        printf("ROUND %2d ", round);
        dump2dcolsBC(label, thevalues, limitrow, limitcol);
    }
}

/*****
*
**/
void dumpvaluesKC(int round, char* label, word8 thevalues[4][MAXBC],
                 int limitrow, int limitcol, bool printflag) {
    if (printflag) {
        printf("ROUND %2d ", round);
        dump2dcolsKC(label, thevalues, limitrow, limitcol);
    }
}

```

```
}

/*****
*
**/
void fillkeyallzeros(word8 thekey[4][MAXKC]) {
    int i, j;
    for (j = 0; j < KC; j++) {
        for (i = 0; i < 4; i++) {
            thekey[i][j] = 0;
        }
    }
}

/*****
*
**/
void fillkeyd2(word8 sk[4][MAXKC]) {
    sk[0][0] = 0x2b;
    sk[1][0] = 0x7e;
    sk[2][0] = 0x15;
    sk[3][0] = 0x16;
    sk[0][1] = 0x28;
    sk[1][1] = 0xae;
    sk[2][1] = 0xd2;
    sk[3][1] = 0xa6;
    sk[0][2] = 0xab;
    sk[1][2] = 0xf7;
    sk[2][2] = 0x15;
    sk[3][2] = 0x88;
    sk[0][3] = 0x09;
    sk[1][3] = 0xcf;
    sk[2][3] = 0x4f;
    sk[3][3] = 0x3c;
}

/*****
*
**/
void fillkeytest(word8 sk[4][MAXKC]) {
    sk[0][0] = 0x2b;
    sk[1][0] = 0x7e;
    sk[2][0] = 0x15;
    sk[3][0] = 0x16;
    sk[0][1] = 0x28;
```

```
    sk[1][1] = 0xae;
    sk[2][1] = 0xd2;
    sk[3][1] = 0xa6;
    sk[0][2] = 0xab;
    sk[1][2] = 0xf7;
    sk[2][2] = 0x15;
    sk[3][2] = 0x88;
    sk[0][3] = 0x09;
    sk[1][3] = 0xcf;
    sk[2][3] = 0x4f;
    sk[3][3] = 0x3c;
}

/*****
*
**/
void filltestd3key(word8 sk[4][MAXKC]) {
    sk[0][0] = 0x00;
    sk[1][0] = 0x00;
    sk[2][0] = 0x00;
    sk[3][0] = 0x00;
    sk[0][1] = 0x00;
    sk[1][1] = 0x00;
    sk[2][1] = 0x00;
    sk[3][1] = 0x00;
    sk[0][2] = 0x00;
    sk[1][2] = 0x00;
    sk[2][2] = 0x00;
    sk[3][2] = 0x00;
    sk[0][3] = 0x00;
    sk[1][3] = 0x00;
    sk[2][3] = 0x00;
    sk[3][3] = 0x00;
}

/*****
*
**/
void filltextallzeros(word8 thetext[4][MAXBC]) {
    int i, j;
    for (j = 0; j < BC; j++) {
        for (i = 0; i < 4; i++) {
            thetext[i][j] = 0;
        }
    }
}
```

```

}

/*****
*
**/
void filltextd2(word8 a[4][MAXBC]) {
    a[0][0] = 0x32;
    a[1][0] = 0x43;
    a[2][0] = 0xf6;
    a[3][0] = 0xa8;
    a[0][1] = 0x88;
    a[1][1] = 0x5a;
    a[2][1] = 0x30;
    a[3][1] = 0x8d;
    a[0][2] = 0x31;
    a[1][2] = 0x31;
    a[2][2] = 0x98;
    a[3][2] = 0xa2;
    a[0][3] = 0xe0;
    a[1][3] = 0x37;
    a[2][3] = 0x07;
    a[3][3] = 0x34;
}

/*****
*
**/
void filltexttest(word8 a[4][MAXBC]) {
    a[0][0] = 0x74; // t
    a[1][0] = 0x68; // h
    a[2][0] = 0x69; // i
    a[3][0] = 0x73; // s
    a[0][1] = 0x20; // blank
    a[1][1] = 0x69; // i
    a[2][1] = 0x73; // s
    a[3][1] = 0x20; // blank
    a[0][2] = 0x74; // t
    a[1][2] = 0x68; // h
    a[2][2] = 0x65; // e
    a[3][2] = 0x20; // blank
    a[0][3] = 0x74; // t
    a[1][3] = 0x65; // e
    a[2][3] = 0x78; // x
    a[3][3] = 0x74; // t
}

```

```
/******  
*  
**/  
char* readstuff(char* filename) {  
    char* text = NULL;  
    size_t linecap = 0;  
    // ssize_t linelen;  
    FILE *fp;  
  
    fp = fopen(filename, "r");  
    getline(&text, &linecap, fp);  
    fclose(fp);  
  
    return(text);  
}
```

Literatur

1. H.O. Yardley, *The American black chamber* (Bobbs-Merrill, 1931), S. 140–171
2. F. Pratt, *Secret and urgent* (Blue Ribbon Books, 1939)
3. J. Daemen, V. Rijmen, *The design of rijndael*, 2. Aufl. (Springer, 2020)